AN INTEGRATED COMPUTER AIDED DESIGN SYSTEM FOR RAILWAY FREIGHT VEHICLES

Volume 1

A thesis submitted for the degree of

Doctor of Philosophy in Mechanical Engineering

in the University of Canterbury

Christchurch, New Zealand

by

D.W. Goddard

University of Canterbury

September 1987

CONTENTS

Volume 1

continued...

CONTENTS (continued)

continued...

CONTENTS (continued)

continued...

CONTENTS (continued)

continued...

CONTENTS (continued)

continued...

continued...

CONTENTS (continued)

continued...

CONTENTS (continued)

LIST OF TABLES

Volume 1

Volume 2

LIST OF FIGURES

Volume 1

continued...

continued...

continued...

continued...

LIST OF FIGURES (continued)

continued...

continued...

continued...

continued...

ABSTRACT

This thesis investigates, with particular reference to New Zealand Railways Corporation, the use of computers in the design of railway freight vehicles. An investigation was carried out into ways of integrating the relevant published works, procedures used in NZR wagon design, and the design and operational experience of NZR staff. The scope of this work did not extend to detailed study of draughting, graphic, and technical office activities as these problems have been and continue to be extensively investigated by other workers.

The results of a literature survey, a survey of NZR Design Office procedures and documentation, and staff interviews are presented. This information was used to produce the functional specification of an integrated computer-based scheme presented in the thesis. The features of a prototype system developed to fulfil some of the more fundamental aspects of the specification are described.

The prototype system used a commercially available relational database management system to store instances of analysis variables, approved project data, catalogue data, and the relationships between instances. The relationships between instances of analysis variables serves to maintain integrity in an active design project and acts as a "computerised notebook". The prototype system offered the wagon designer substantial flexibility in the use of data and over the control of execution. The author suggests that extensions and changes can be easily implemented in the prototype system.

Experience with the use of this prototype system is presented and views regarding its proposed development are expressed. The feasibility

of much of the specification was demonstrated and an assessment of probable benefits and costs showed that such an integrated scheme would overall contribute more than other approaches to the corporate objectives.

CHAPTER I

INTRODUCTION

Railways the world over are facing increasing competition within the transport industry. The use of computer systems in the design of railway vehicles could potentially lead to the provision of better services at cheaper cost than presently available. This study investigates, with particular reference to New Zealand Railways Corporation, the use of computers in the design of railway freight vehicles.

1. RAILWAY FREIGHT TRANSPORT

In this section, railway freight transport is defined, organisations involved in railway freight transport and their objectives are briefly discussed, and wagon design within the context of these systems is introduced.

(1) Definition of Railway Freight Transport

Railway transport can be characterised by twin steel rails on a track bed approximately at ground level, and by the utilisation of steel wheel to rail contact as the means of generating guidance and levitation. Typically goods carrying vehicles run in articulated trains with motive power concentrated in specific units but with braking capability distributed through the train. Locomotives apply the tractive effort, and supply and control the power for the braking

system, while freight wagons are vehicles designed to solely transport a specific freight commodity or range of commodities.

(2) Introduction to Railway Organisations

Railway organisations are generally large systems with a high degree of self sufficiency; systems which provide their own guideway, vehicles, terminals, control systems, and maintenance facilities. The provision of these integrated systems requires large capital investment. The extent of this investment dictates gradual introduction of compatible changes within the railway network in response to stimuli. This is one of the reasons why technological advance in the railway transport industry has been evolutionary throughout its 150 years of existence.

In 1984 the New Zealand Railways Corporation (NZR) earned $642 million with assets with a book value of $879 million. Freight or goods traffic generated 64% of this revenue. The major revenue earning commodities included products from forests ($50.4 million), coal ($26.5 million), meat ($29.7 million), wool ($12.2 million), and petroleum products ($8.4 million). Total revenue goods tonnage hauled was 10.6 million tonnes and the total gross goods tonne kilometres was 7,950 million. The average revenue earning goods train (gross) load and average speed were 589 tonnes and 30.5 km/h respectively. The average goods haul was 298 km. The fleet of vehicles included 473 locomotives, 18348 two axled (or four wheeled) wagons, 6200 four axled (or bogie) wagons and 1560 containers. There was 4273 km of 1.067 m gauge track open for traffic. A total of 19747 staff were employed.

(3) Fundamental Advantages of Railway Transport

The large investment in railway systems is principally encouraged by the low power/load ratios required which enable large quantities of goods to be hauled at low unit variable costs for fuel and wages compared to highway carriers. For example, in NZR a 1,425 horsepower DC class locomotive manned by a driver and assistant can haul around 650 tonnes up a 1 in 60 grade. Thus it is often held that railways are best for long distance bulk overland transport, but changes in fuel, bitumen plant costs and the like are continually forcing revisions in the definitions of "long" and "bulk". (However on the debit side are train delays; marshalling labour costs and damage; and terminal loading/unloading time.) The advantages to society at large of rail transport (when compared to road transport) can include a reduction in noise and air pollution, a reduction in danger of spillage or leakage of hazardous substances, and a reduction in road congestion and road accidents. These considerations are particularly important in cities and residential areas where road transport is less confined and disciplined than rail.

(4) Objectives and Performance Measures of a Railway Organisation

The railway industry has aims of an enduring nature - the aims are about survival through service to customers. The transport service provided by railways produces nothing (in the make or manufacture sense of the word) but is part of an industry (that is, the transport industry) essential to the wealth creating process. People and organisations are willing to pay the cost of transport if value is

added to goods when they have been moved to a specified geographical location. The output of a railway can be measured in tonne - km.

For NZR the New Zealand Railways Corporation Act (1981) sets out the functions and objectives of the Corporation. In part they are:

"(a) To establish, maintain, and operate, or otherwise arrange for, safe and efficient rail freight and passenger transport services within New Zealand."(New Zealand Railways Corporation Act, 1981, clause 12(1))

Railway freight service can be characterised by levels of safety, damage to goods, reliability and social acceptability, by transit times, frequency of transit, timing of service and accessibility of service, and by speed of response to changing market demands. Some measures of these qualities are:

- number of fatal and injuring accidents
- number of loss, theft, and damage claims
- dollars per claim
- number of complaints
- average number of delays per service
- average train delay
- average transit time
- average terminal handling time
- number of regular and special services per week
- number of terminals.

However the quality of the service must be balanced against what the customer is charged so the service is an attractive proposition compared to what other commercial transport organisations offer. Furthermore the price charged for the service compared to the cost of providing this

service must be such that a return on investment is provided, which suggests more measures of performance:

- cost per tonne - kilometre
- average cost of terminal handling.

NZR has the statutory responsibility of providing a satisfactory return to its shareholders - who are the people of New Zealand. Clause 12(1) paragraph (e) of the New Zealand Railways Corporation Act states:

"(e) To endeavour to carry on the operations of the Corporation in such a way that revenue exceeds costs, including interest and depreciation; and to provide for a return on capital that may be specified from time to time by the Minister of Finance."

The corporate objectives for a particular organisation consist of a time varying set of ranked formal and informal objectives. It is management's task to formulate from its perception of the corporate objectives and information, requests, ideas, and so forth it receives, the sub-objectives and goals for the various organisational groups. For the wagon design group, these might be, for example, higher axle loads, reduction in tare weight, installation of automatic couplers, increased train size, higher operating speeds, and removal of the four wheel wagon fleet.

(5) Introduction to Wagon Design Subsystem

A necessary function for the railway industry is wagon design. In NZR the Mechanical Branch is responsible for providing and maintaining freight wagons. The majority of freight wagons have been designed in the Design Office of the Mechanical Branch, the remainder being supplied by design/manufacture companies.

In subsequent chapters two aspects of wagon design will be expanded and discussed more fully. They are: wagons and their environment in the acquisition and operation stages of their life cycle; and the wagon design system and its environment.

For NZR, the wagon design system is the elements within the Design Office which contribute to the wagon design function. In its environment are the other elements within NZR (for example, the Way and Works Branch who provide the guideway and associated bridges, tunnels, and track structures; and the Chief Mechanical Engineer - the Manager of the Mechanical Branch - who approves the commitment of resources to major wagon design system projects), or outside of NZR (for example, customers and wagon design/manufacture contractors) with which the wagon design system interacts.

The primary function, ascribed by railway management, of the wagon design system is the production of specifications for the conceptual design, design analysis, detail design (and design documentation), or for manufacture of new or revised wagons or their components. The involvement of the wagon design system with a particular wagon design does not finish with the production of manufacturing instructions and working drawings, as throughout the acquisition and operation phases the wagon design system issues modification instructions to correct for problems or to improve performance.

Cost and fitness of purpose in the acquisition and operation stages of the wagon life cycle are wagon attributes directly influenced by the wagon design system. A secondary function of the wagon design system is to assess from wagon performance, and from related events and actions

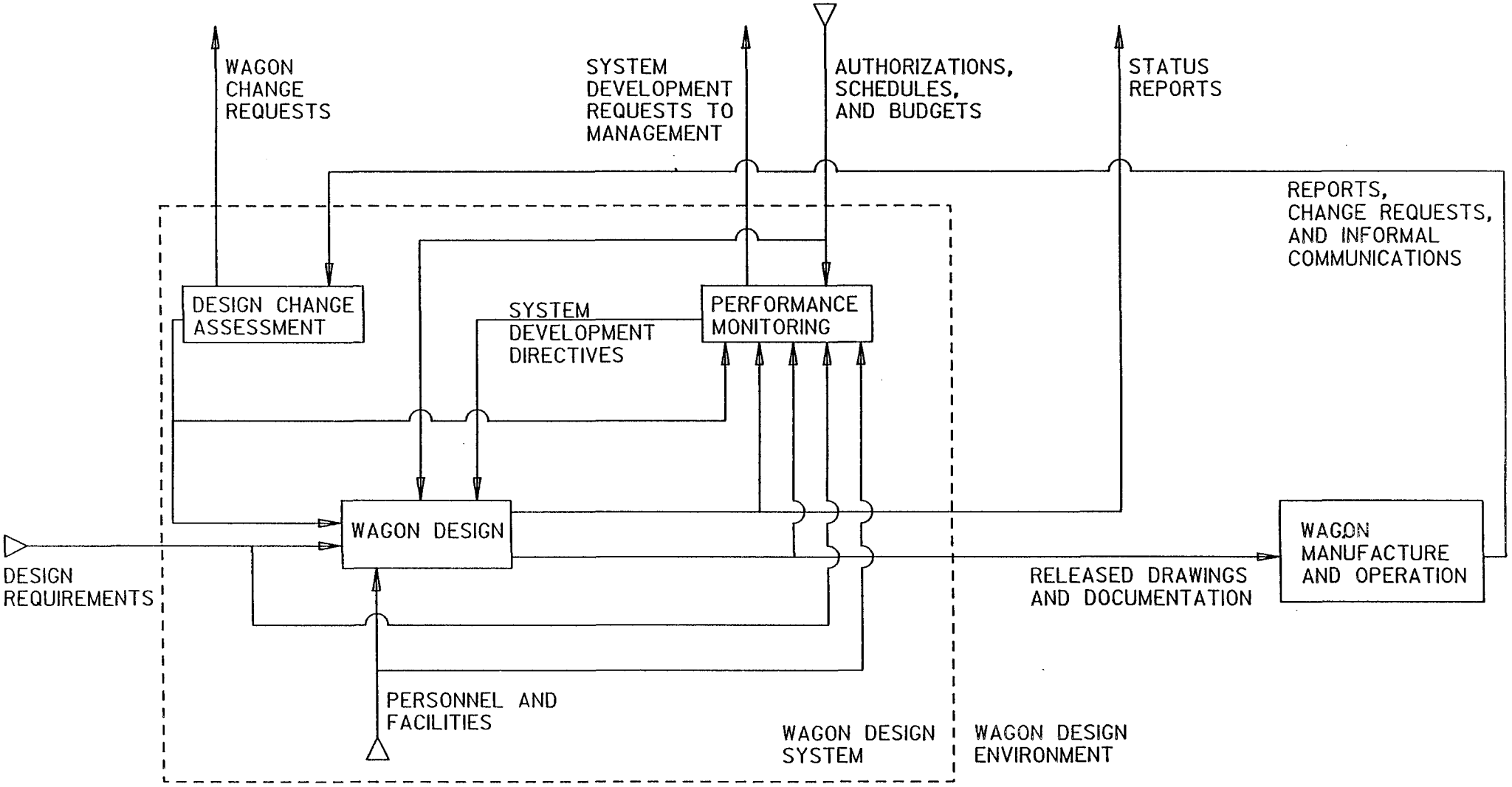


Figure 1. Functions of the Wagon Design System

in its own environment the need for new or modified designs, and to communicate identified needs and make requests for resources to the authorising elements in its environment (refer fig. 1).

The cost and timeliness of responses to the requests by management for new or modified wagons are important attributes of the wagon design system's activities. There is another secondary function which is to maintain or improve these wagon design system performance attributes.

2. USE OF COMPUTERS IN DESIGN ORGANISATIONS

This section outlines reasons for introducing computers into design organisations and describes some of the problems involved. A computer system is that collection of computing equipment, amendable sequences of coded instructions ("software") and operating procedures which can be accessed by an authorised person to run jobs.

(1) Benefits In Introducing Computers into Design Organisations

(a) Computers in Engineering. Computers have successfully been applied to many engineering activities including cost estimating; life-cycle costing; investment appraisal; preparation and editing of specifications and reports; document storage, retrieval and release control; technical illustrating; charting and graph preparation; calculation and analysis (structural, kinematic, mass and section properties); clearance/interference studies; visualisation; drafting; data reduction and statistical analysis; numerical control tool cutter path generation, editing and simulation; production and maintenance

planning and control; project management; plant layout; calendar management; distribution and management of notices and mail.

For although present day computing systems have but five basic abilities - input of data, processing, output of information, storage of information, and communication of information - they have successfully been applied in many fields because they perform these functions with speed, tirelessness, consistency and accuracy.

(b) Computers in Design. Computer-aided design is a process which uses computers to assist in the creation or modification of a design. Typically the computer provides interactive assistance to a person working at a console containing at least one graphics display.

The use of computers in design is a progression in the development of support systems (for example, pencil and paper, and slide rules) that augment the limitations in human intelligence (for example, the limited short term memory, inabilities to predict the consequences of assumptions, or to follow long chains of reasoning steps, or to control in a logical manner complex situations with many non-deterministic interactions between components of a system).

Computer systems can provide: more automated (and often faster) information processing which frees the human designer from the clerical and arithmetical tasks; more assistance to exploit the strong parts of the human information processing system such as summarising, visualisation, context-sensitive views filtering out irrelevant detail from information structures, browsing consisting of movement around complex information structures and focus on areas of interest; large shareable external memory capacity for precise description of the design alternatives, the design path, and design alterations; more "self-

knowledge" on how and where to find information in the system and how to process it; and more management control over methods, components, design documentation, and synchronisation of updating processes, information release, and communication. The goal of much of the development in these areas has been the transfer of work to the machine so the human can say what is required to be done and the machine decides how to perform the task.

Examples of automation of mathematically described problems include: mass property calculation; structural analysis; rule checking; dimensioning; geometry copy, rotate and placement; and generation of part geometry using standard features and variable parameters. Modifiable models on computer systems allow the rapid production of a large set of alternative designs (prototypes), cheap enough to be thrown away yet detailed enough to be tested and argued about. More sophisticated simulation and evaluation is possible with these models.

Automation of clerical processes include: recording of resource usage against jobs; monitoring work in progress; alteration recording and communication of component changes; and propagation of changes.

Assistance to the human information processor comes in many forms, for example: colour coded stress distributions; deflection plots; views of 3D geometry from any angle; "levels"; many documentation views - parts lists, exploded views, dimensioned working drawings - from the one information store; retrieval of the same or similar designs to any scale by specifying any combination of attributes; and maintenance of an agenda of things to do. Features such as these can provide improved design insights and a wider range of strategies for the design process.

The external memory aid can share information in a controlled manner

between, say, design product structuring and production planning, between concept geometry and finite element analysis, or approved product geometry and numerical control programming. Not only does it allow controlled reference to the work of others (individuals or application programs) but also controlled reference to the status of the design at some earlier time. Duplication of effort and transcription errors are reduced and updates are available to all the moment they are released.

For their part in the computer- aided design system, humans contribute goals, integrate different knowledge sources, apply imagination and intuition (and these processes, that is, intuitive and imaginative are predominant in complex holistic situations for the unaided designer), use common sense knowledge and solve problems by analogy (by any measure, still an essential contribution).

Many benefits to an organisation, some more amenable to measurement than others, have been claimed attributable to the introduction of computers into their design system.

Primrose and others (1985) have compiled a list of potential benefit items, each defined in such a way that it can be quantified and danger of duplication is avoided. They are:

(i) Design Office Savings

- Reduce the number of existing designers by increasing productivity.

- Avoid recruiting additional designers by increasing productivity.

- Reduce clerical labour in Design Office by increasing productivity.

- Reduce or avoid subcontract design work by increasing productivity.

- Take on subcontract work by increasing productivity.

- Eliminate physical model and prototypes by use of 3D design.

- Reduce outside graphic design work (for marketing, service departments, publicity, etc.) with increased productivity and technical capability.

(ii) Increased Sales from Reduced Delivery Times

- Reduce design/documentation time for customer orders by increasing design/documentation creation and modification speed.

- Reduce production time with improved designs and documentation (e.g. easier assembly).

- Reduce production delays due to incorrect ordering of components by providing a consistent and accurate set of documents.

(iii) Increased Sales from Other Causes

- Company can quote reliable delivery dates with reduced errors and improved estimating capability.

- Faster and better presented quotations.

- Company image improved by having CAD.

- Orders would be lost if company did not have CAD facility to prevent "excessive" quotations.

- New products can be introduced more quickly leading to earlier overhead recovery and "first in the market" appeal.

(iv) Reduced Stock Levels

- Reduce work in progress due to reductions in production lead times which in turn is caused by improved designs and

documentation.

- Reduce finished stocks by increasing component standardisation.

- Reduce ordering of unwanted components by improved accuracy and consistency in documentation.

(v) Reduced Production Costs

- Reduce production costs with improved designs and documentation (e.g. easier assembly).

- Reduce scrap and rework (including that on jigs, tools and materials) by reducing design and documentation errors.

- Improved production efficiency by eliminating "stock outs" due to improved accuracy and consistency in design documentation.

- Larger batch sizes and economies of scale due to component standardisation.

- Reduce production and material costs due to design optimisation.

(vi) Cost Control

- Unprofitable orders eliminated by improved estimating.

- Company internal cost control improved by providing more accurate estimate with which to identify cost variances.

(vii) CAM Link

- The purchase of separate systems for NC programming avoided by CAD.

- Reduced NC programming costs by linking programming to CAD.

- CAD aids company-wide information system

- CAD avoids the need for expenditure on, for example,

expanding office space.

Other potential benefits seemingly omitted from this loosely defined list yet common in articles on the matter are:

- Increased sales due to reduced production delays when design/documentation changes are necessary to meet unforeseen construction requirements or to correct deficiencies.

- Increased sales due to improved product quality, fitness of purpose, and better satisfaction of customer needs. (For NZR, new wagons are not sold to the operating group, but this factor could cover improved revenues and reduced maintenance and operating costs - for example, reduced wagon weight reducing fuel costs and track damage.)

- Reduced warranty claims due to improved product quality and reduced exposure of the company to technical risk.

- Reduced orders lost if company did not have CAD facility to receive and transmit design data in electronic form.

- Reduced expenditure on attracting/keeping top personnel due to CAD technology leading to enhancement of their skills; improvement in pay and status, and greater job satisfaction.

It has been noted by some observers that consideration of the introduction of computers has presented an opportunity to critically review existing practices and organisational structure, the review in itself generating benefits such as listed above.

(2) Disadvantages of Computers in Design Organisations

The first group of disadvantages relate to a function (as seen through the eyes of an employee) of a design system, that function being

the provision of satisfying employment. These problem areas have implications for the organisation itself.

Cooley (1976) has compared the recent advances in the computerisation of intellectual work (in particular, CAD) with the mechanisation of manual work in the Industrial Revolution and has suggested that there will be increasingly:

(a) Subordination of the designer to the computer. Shift work or systematic overtime to counter the rate of obsolescence of the machine (and the shorter life of fixed capital) and large capital investment required. Limitations on the creativity of the designer imposed by standard routines and optimisation programs. Capture of the workers knowledge (which give the workers bargaining power) into a machine usable form. Domination of the subjective value judgements of the designer by the "objective" decisions of the system. Alienation of the designer from his work and his product. Abstraction of the design activity from the real world.

(b) A fragmentation of design skills (over-specialisation) with a loss of panoramic view, and the introduction of "scientific management" to the extent of measuring the rate of performing intellectual work. De-skilling the design function. Increased work tempo as the designer is paced by the computer. Increased physical and mental stress.

(c) Loss of control over one's work environment.

(d) Growing job insecurity, particularly for those less adaptable.

(e) Knowledge obsolescence and little opportunity for experienced workers to develop new skills and capabilities.

(f) The proletarianisation of the design community and consequent increase in industrial militancy.

In the second group of disadvantages, there are potential costs (other than initial development and running costs for the computer system) directly affecting the organisation's performance.

If fewer staff are, through productivity improvements, producing the work previously performed by four to ten times the number of workers, the organisation has then placed greater dependence on individual members of staff. The knowledge of the job is vested in fewer people increasing the susceptibility of the organisation to impaired efficiency when staff leave. Also greater reliance is concentrated in the communication, processing, storage, output and input machinery, again increasing the vulnerability of the organisation.

There is also the danger of developing an aid (or tool) for a task and finding later that one's view of the task is restricted (consciously or unconsciously) to that which is achievable with that aid. This dependence could be attributable to pride in the aid, laziness, unwillingness to challenge the "computer", forgetting the assumptions built into the aid, and/or forgetting the simpler manual engineering methods.

Whereas the experienced designer has wide ranging knowledge of past failures, the computer program has only those models specified by its human creators. The availability of software can give the illusion of power over the complexity of nature and can lead to designers taking on jobs that are at best on the fringe of their expertise. Then the designer cannot be relied upon to ask the right questions and exercise good judgement about failure modes and likely situations. The answers from the computer can only at best be as accurate as the input is known.

The output from computer programs can promote the drawing of unwarranted inferences and the formation of intuitions that do not reflect physical reality. Computer programs are a simplification of reality. Graphic displays in particular can be subtly yet systematically misleading.

These potential dangers point to the need for a considered application of computers to design.

3. NATURE AND SCOPE OF THE INVESTIGATION

This section covers what this thesis is about and why it was undertaken.

(1) Scope

This thesis investigates the wagon design system and its environment as defined in section 1 of this chapter, with the exception of detailed study of draughting, graphics, and technical office activities. These areas are excluded from detailed consideration (albeit a rather arbitrary boundary) because they are significant topics in themselves whose problems are not peculiar to computer-aided design of railway wagons. These problems are being investigated and documented elsewhere. Extensive efforts by other organisations and individual workers have produced many commercially supported packages, but only recently in the graphics field have standards emerged which offer increased portability. It is also necessary to restrict the scope to take into account the limited nature of a Ph.D. study and the resources available at the University of Canterbury.

The term design, then in this thesis, encompasses synthesis and analysis activities but excludes project management, technical document composition, draughting, and generation and display of geometric models. However all the activities of the wagon design system and the interactions with its environment were studied to some extent, from the receipt of change requests and development directives through to the production of drawings and manufacturing instructions. From this research a functional specification of computer-aids for the wagon design system and a prototype system to test out aspects of the specification were produced.

This thesis does not review or select commercially available computer-aided design systems or equipment (although the system specifications do indirectly place some restrictions) because these activities are logically later in the design and implementation of the computer-aided system. (Although of a secondary nature, the reasons given for excluding graphics and draughting apply here also.) However some features of commercial systems have been discussed to illustrate a point or where they are particularly innovative with regard to this thesis.

(2) Purpose

(a) The Problem. The problem this thesis is concerned with may be posed as: the identification of how to make use of computers in wagon design so as to increase New Zealand Railways Corporation's performance in attainment of its objectives.

However, the problem re-examined in light of the normal constraints in time and resources applied to Ph.D. study becomes one of specifying

the arrangement of existing components (the computer hardware, operating systems, programming languages, database management systems, analysis and optimisation packages, and so on) so as to form a computerised wagon design system that will aid the attainment of NZR's objectives.

Although a number of application programs exist for wagon design that provide benefits in many of the areas listed in section 2 of this chapter (as will become evident to the reader in subsequent chapters), they suffer from a number of deficiencies when viewed as a system of computer-aids.

The disjointed application programs are developed and maintained with duplication of effort. Their means of communicating with the wagon designer reflect the different styles of the originators, requiring considerable effort on the part of the wagon designer if he is using a number of programs on a casual basis.

The design calculations are recorded in paper form in notebooks and program printouts or in disjointed and often incompatible computer-based application files. (A file is a logical container for data defined to the computer system which has certain attributes and may be divided physically between various devices and machines that comprise the computer system.) Where many computer-based systems can adequately store information and communicate data (if it has been defined and assembled into a file) to individuals possibly in different geographical locations, most of the application file oriented systems (that is non-integrated) are not good at recording the relationships between different versions of an application program's data used by different applications, nor are they good at facilitating the communication of information between applications. The transfer of data manually or by

means of a computer system to other applications is usually a time consuming and error prone process. In common with manual records on products and existing designs (such as drawings, catalogues, and handbooks) data stored with these non-integrated computer-aids lack co-ordination and flexible access methods; they are time consuming to search; uncontrolled duplication of data is widespread which provides fertile ground for inconsistencies. The manual management and updating of such a file oriented system places limitations on speed, accuracy and consistency.

An integrated system, rationally and systematically designed with integral sharing and communication of data will not have these problems.

The author did not find any attempts to produce an integrated wagon design scheme, in either the literature surveyed or during the author's visits to several organisations involved in wagon design in the United States, Great Britain and Europe. There have been attempts at integrated design in other areas, some of which are discussed later in the thesis. The basic components of an integrated design scheme (the operating systems, database management systems, data transfer standards, the man machine interface, and so on) are in a much more developed state than they were for the earlier attempts in the 60's and 70's. This should enhance the probability of success for the scheme.

Although in this study the size of the operation is small by world standards, and the mix of activities is probably different from other design groups, the author suggests the results will be of interest to other workers in the more general field of computer-aided design.

In an over supplied transport market NZR is presently retrenching and reducing its wagon fleet and the number of new wagons built.

However in the opinion of the author the need for such a system as proposed is no less pressing in this environment (deregulated transport industry since 1981) than in an environment of prosperity, quite the contrary. To operate successfully in a more competitive market modern wagons better suited to their purpose (which includes making a profit with them) are required to retain existing traffic. In addition, to capture new markets and customers, NZR will have to explore the alternatives quickly and make a speedy response to requests. The risk of failure in the economic sense is now much higher.

(b) The Objective. The original statement defining the study was: "The aim of the proposed project would be to produce for NZR an integrated design scheme based upon published experimental and theoretical research results of other workers on structures, wheel-rail interactions and the like, and upon the design and operational experience of NZR staff. It would also involve sorting out which design parameters may be determined logically and which are still dependent upon experience or intuition. The logical procedures could be programmed for a computer, to reduce the time involved and to give design managers better control over the design process. Such a scheme should enable NZR to design, in a shorter time than at present, wagons that would be more efficient and/or more reliable and/or more enduring, according to operational requirements." This statement was interpreted by the author to mean the provision of at least a functional specification of an integrated design scheme; a scheme to assist in the attainment of NZR objectives.

This thesis will have to:

(i) Demonstrate that the proposal is technically possible from a computing technology point of view.

(ii) Demonstrate the operational and implementation feasibility of the man machine system within the proposed working environment. The impact of the computing system on people (and vice versa) will have to be assessed.

(iii) Assess the probable benefits and costs.

To aid in this assessment a prototype implementation was envisaged.

(c) The Hypothesis. This thesis, then, puts forward the hypothesis that a wagon design system (such as can be found in NZR and in such an environment as can also be found in NZR) that is supported by the integrated system specified in chapters X to XIII contributes more to the attainment of the corporate objectives than the same wagon design system without an integrated system.

(3) Outline

The objective of this thesis has been presented, but in order to break down the objective into more detailed requirements with a view to implementing some proposal, we must study the properties of wagons and their environment, and of the wagon design system and its environment - the wagon designers' characteristics and needs, the organisation and the activities. The study of the wagon properties, the present wagon design system and its environment, and other published works on aspects of wagon design is contained in chapters II to VIII. Chapter IX provides a general introduction to relevant computing technologies. Chapters X to XIII cover the system specification, while in chapter XIV some previous attempts at integration of computer-based interactive systems are

discussed. Chapters XV through XIX deal with the implementation of an experimental system which meets a number of the more fundamental aspects of the specification including the maintenance of relationships between multiple instances of variables and flexible, interruptable processing of data. The conclusions and comments on the prototype system are presented in chapter XX.

4. REFERENCES

COOLEY, M.J.E. (1976) CAD : a trade union viewpoint. Proc. CAD 76 - International Conference on Computers in Engineering and Building Design, London. Guildford, IPC Press. p.308 - 312.

New Zealand Railways Corporation Act 1981

PRIMROSE, P.L. and others. (1985) Identifying and Quantifying the Company-wide Benefits of CAD Within the Structure of a Comprehensive Investment Program. Computer-aided Design, 17(1) : 3-8.

CHAPTER II

THE PRESENT WAGON DESIGN SYSTEM

1. INTRODUCTION TO THE SURVEY OF WAGON DESIGN

(1) Scope

This chapter and the following six chapters document NZR's existing wagon design system and wagons, supplemented with published information on overseas vehicle design. These chapters cover the structure of the wagon design system and the process of wagon design, the variations in the geometrical and material properties of wagons and of wagon components, and the functions and performance of wagons and wagon components.

Theoretical and experimental aspects are more predominant than the intuitive aspects of wagon design as by definition what is intuitive is not explicit and only by research can it possibly become so.

Not only is it beyond the scope of this thesis to argue one design procedure or one design artefact against another, it would also exceed the author's authority to prescribe design methods or criteria for NZR wagon designers. Therefore a range of designs and proven procedures of varying sophistication are described. Advancing the knowledge of the individual fields of wagon behaviour has been left to the researchers working in those fields, but brief mention is made of some recent changes. These advances will inevitably be reflected in new or modified design procedures and wagons.

(2) Purpose

There are a good many sources of information not referenced and discussed due to limitations of space and time and there are doubtless many of which the author is not aware. The author has attempted to illustrate in sufficient detail the processes involved in or expected to be used in wagon design. This sample acts as a basis for the computer-aided wagon design system specification presented later in the thesis. Therefore what is contained in the following chapters is not comprehensive or detailed enough to fully describe any particular wagon or to permit any wagon to be fully designed.

(3) Methods

Information on NZR's wagon design system was gained through interviews with staff of the Mechanical Branch's Design Office, and from study of their design reports, test and service reports, referenced texts, standards, manuals of recommended practice, and working drawings and specifications of existing wagons. The international scene was studied by literature survey followed by visits to several railway vehicle design organisations.

2. THE WAGON DESIGN SYSTEM STRUCTURE

The Mechanical Branch's Design Office is located in one open plan office in the Wellington Railway Station. In addition to wagon design it also performs locomotive, passenger stock, container, and mechanical appliance design.

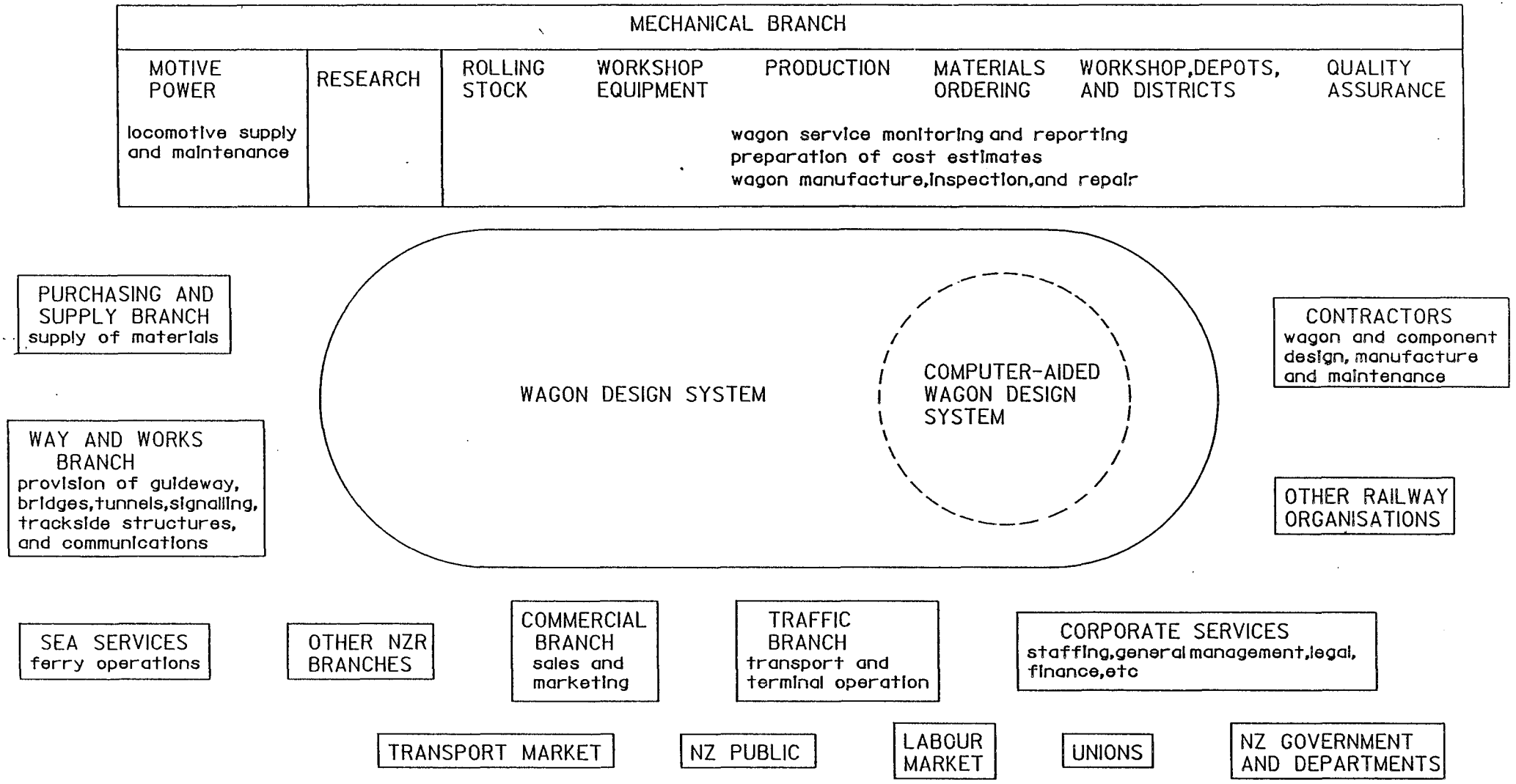


Figure 2. NZR Wagon Design System and Environment

(1) Design Office Staff

The Design Office is headed by the Designing Engineer, who is assisted by the Mechanical Appliance Engineer and the Assistant Design Engineer. These managerial positions are filled by experienced registered engineers. Many of the Design Office staff have completed trade apprenticeships and some have technician qualifications (such as NZCE) and others have university degrees in engineering. Typically over the last several years two or three of these Engineering Officers have had 15 or more years experience in the Design Office. Of the staff who have been in the Design Office less than five years and who from time to time may perform wagon design (draughting, modifications and so on), there has been on average in recent years six tradesmen or school leavers, two recent engineering graduates, and two graduates nearing registration. The total number of staff ranges about twenty. Some individuals are recognised as having expertise in particular fields, but most have wide ranging experience.

The wagon designer is an individual who performs wagon design within the wagon design system.

(2) The Environment of the Wagon Design System

Figure 2 shows the wagon design system and the major systems considered to be its environment.

In the environment of the wagon design system are the other organisational groups within the Mechanical Branch, which are responsible for wagon manufacture, maintenance, quality assurance and service monitoring, financing and planning, and for motive power (or locomotive) supply and maintenance. The head of the Branch management

group is the Chief Mechanical Engineer (CME). The Designing Engineer reports directly to the CME. The wagon design system, in the course of wagon design or modification, also interacts with other Branches in NZR such as the Way and Works Branch (which provides the guideway or permanent way and associated bridges and tunnels, signalling and trackside structures), the Traffic Branch (which assign wagons to transport tasks and performs the transport and terminal operation), Sea Services (Cook Strait ferry operations between North Island and South Island), the Commercial Branch (the sales and marketing team), Purchasing and Supply Branch, Corporate Services such as Finance and Accounts, Legal, Planning, Personnel, and General (or corporate) Management. Outside the Corporation the wagon design system deals with customers, contractors, material and part suppliers, railway labour unions, designer labour market, engineering groups, other organisations involved in the railway industry, NZ Government departments, and public pressure groups.

3. IDENTIFICATION AND DEFINITION OF A NEED

(1) Initiation of a Design Project

Design activities can originate from the acquisition and operation stages in the wagon life cycle, that is:

- declaration of a need;
- prototype manufacture and test;
- production and assembly;
- inservice and maintenance monitoring.

Communication on these matters to the Mechanical Branch may be by way of the telephone, in specific reports, but usually by letter. All

written correspondence regarding a particular topic is placed in a file covering that topic. The initiator may be other sections of the Mechanical Branch as in the case of maintenance reports, test reports or requests for wagons to carry, for example, wheelsets; the Way and Works Branch may require wagons to carry, for example, ballast; or more typically a customer's request for wagon(s) to carry a specified load will be forwarded by the Traffic and Commercial Branches if there are no wagons available that would provide satisfactory performance for that load. It may be a perceived requirement (perceived by Commercial Branch) rather than a direct request by the customer.

New or modified transport tasks will require specified payloads, defined by weight, dimensions and other physical properties, to be carried on specified routes. Discussions with the potential customer may be necessary to clarify or refine the initial statement of needs.

(2) Development of a Proposal and Performance Specification

The Branches involved (for example, Mechanical, Way and Works, and Traffic Branches) determine operations proposals (that is, number of trains/day, total number of wagons and locomotives required, and so on) together with project costings and schedules.

If a line is to be upgraded or constructed a number of route options (of various grades, curvatures, and so forth) may exist and it may be possible to have maximum axle loads, minimum axle spacing (avoiding over-stressing of bridge structures) and loading gauge (or maximum vehicle cross-sectional outline) which are significantly different from the rest of the rail network. Increasing axle loads, speeds, trailing tonnage and improving operation of terminals increases

the utilisation of rolling stock and track (the big capital investment items) but can incur higher maintenance charges. Reducing the number of wagons for a fixed trailing tonnage by increasing axle load also leads to fewer wagons to be maintained, a reduction in capital costs in wagons and terminals, a reduction in rolling resistance per gross tonne, and an increase in payload per gross tonne. The maximum train loads are governed by locomotive characteristics, draw gear and brake limitations and route characteristics. Speeds are limited due to considerations of safety, track damage, vehicle and lading damage, train dynamics, and operational practices.

Wagon design aims at a design that will be economic to manufacture, will perform efficiently, reliably and safely, will be easily maintained at low cost and minimum stand down time, and will endure for an economic life. The measurement of performance involves a number of factors including procurement costs, component failure rates, failure modes, wear-out life, repair and service costs, inspection costs, scrap values, inventory costs, failure detection probabilities, probabilities of repair, inservice component failure costs (especially train delay and derailment costs), load/unload costs, fuel costs, locomotive costs, permanent way deterioration, wagon space and payload utilisation. (A failure is the loss or degradation beyond acceptable limits of a function performed by the artefact, and a failure mode is the symptoms that characterise a particular form of failure.) The methodology for analysing and comparing alternatives needs a model relating the various elements in the life cycle (for example, inspection and operation practices, environment control, component design) to component service performance. In reality such comprehensive understanding does not

exist, and much freedom is left to the designer regarding which performance characteristics are to be focused on. There are a number of generally accepted qualitative criteria, and others may be specified by management from time to time.

The production and operating costs of a wagon design are largely determined by the designer but the cost of the design work is typically only of the order of the production cost of one wagon.

Standardisation of wagon designs and/or components can enable bulk orders to be placed and a reduction in unit first costs. Component standardisation and the ability to interchange can ease maintenance and increase availability.

Aerodynamic forces are insignificant in stressing, brake and ride performance calculations at present NZR freight train speeds, although aerodynamic drag and overturning moments induced by side wind can be problems. Reduction in train resistance to save energy costs has been discussed by Muhlenberg (1978).

Weight savings can result in additional revenue earning load, and reductions in operating and maintenance costs and in capital costs particularly with highly utilised wagons.But small changes in wagon tare (that is, less than three or four percent) may have little effect on locomotive operating costs because they would not enable operation at a lower throttle notch or with one less locomotive unit.

Within the wagon design system the payload and size of the vehicle is influenced by such factors as lading density; maximum axle load and load gauge constraints; estimated tare weight; limitations on height of centre of gravity, and on vehicle length and wheelbase (for structural

strength and traversing curves); lading containment and location, and loading/discharge considerations (for example, hopper side and end plate inclination or container lengths and widths). As wagons have a service life of 20 to 40 years likely future service conditions as well as the present must be considered.

Depending on the project's time constraints, preliminary or detail design activities are performed to develop a selection of wagon designs for the project.

The business plan (which contains the project schedule, specifications and justification) is submitted to management for approval, and if they decide to proceed, management will authorise the proposal, perhaps in revised form. At any stage in the design process, management may decide not to proceed with the project. Management will also respond to the customers' interest with information such as rates for transport and land based facilities needed.

4. PRELIMINARY WAGON DESIGN

There have been numerous types of wagons built in New Zealand and overseas and knowledge of these designs may suggest to the designer a suitable concept(s) for the problem at hand.

(1) Considerations in the Design of Special Purpose Wagons

When warranted by the scale of traffic (large enough to ensure high utilisation) special purpose wagons are built to achieve secure and economic handling. Although empty meterage may be high, they offer greater productivity with a high probability of at least one fully

loaded journey for each empty journey, potentially quicker turnaround and savings in manhandling with automation of loading/unloading. For large amounts of bulk freight moving regularly over a route, railways can offer trains dedicated solely to that traffic (that is, unit trains) where marshalling and shunting can be kept to a minimum at the terminals cutting costs and easing scheduling.

Specialised wagons have been designed and built for the transport of specific large objects such as cars, pipes, logs, ISO containers (possibly refrigerated), road semi-trailers, rolled coils of sheeting, newsprint, livestock and so on. Wagons built for goods in bulk include those employing a flow of air (to emulsify goods in fine powder form, for example, flour and cement) to discharge a specialised tank (which may be removable); tank wagons for conveyance of liquid or gaseous freight sealed to prevent leakage and contamination (again the tank maybe easily removed); and hopper wagons for the transportation of goods in granular or aggregate form.

(a) Tank Wagons. Tank wagon designs vary, with regard to:

- tank volume and product density;
- nature or lining of the walls for corrosion resistance (if containing, for example, caustic soda, sulphuric acid);
- resistance to pressure (vapour, temperature and surge generated);
- method of unloading (gravity, pumping, gas pressure);
- safety devices (automatic shut-off valves, internal closing devices, and so on);
- temperature of the product (liquefied gases, refrigerated foodstuffs, heated viscous products).

The majority of these substances are classified as dangerous in that they are flammable, toxic, corrosive or possess some other property which is considered harmful to human life and its environment. Consequently design, maintenance and operation of wagons carrying dangerous goods are constrained by NZR rules, regulations and instructions, national laws, and international codes and standards.

For most NZR tank wagons, the tank design has been contracted out to other firms and the design of the underframe and running gear has been done by the Design Office.

(b) Wagons for Granular or Aggregate Goods. Considerations in the design of wagons for gravity unloading of bulk granular goods (self clearing bottom discharge) included:

- the need to close loading apertures if goods require weather proofing;
- whether or not a means of adjusting and controlling the discharge flow is required;
- the use of bin flow theory in the design of hopper shape for clean, unaided loading and discharge of payload;
- the height of unloading apertures, chosen for discharge into a fixed installation or portable unloading device (for example, portable conveyors, spiral appliances);
- unloading apertures along the centre of the wagon for discharge between the rails, on either side for side discharge, or unloading apertures across the wagon.

Another type of hopper wagon designed for tippler discharge facilities is unloaded by rotating the wagon about its longitudinal axis.

(c) Wagons for Pallets, Rolls and Drums. Other specialised wagons have been developed for palletised goods, drums, and so on (for example, rolls of newsprint; palletised cans, packets or bottles; perishable foodstuffs requiring temperature/humidity control and/or non-tainting linings). Handling techniques for these goods (such as lift truck, hand pallet trucks, cranes) require convenient load access as provided in covered wagons with full height doors or sheets that slide over the whole wagon length; flat wagons with high ends; and covered wagons with folding or sliding rooves. As these types of goods may be susceptible to damage, devices to reduce breakage and tearing are often used and care must be taken in the provision of interior surfaces. Some goods demand volumetric capacity to obtain a useful payload.

(2) General Traffic Wagons

However the majority of the NZR wagon fleet are the covered box, high sided and/or ended, and platform (or flat) wagons in general service, whose design features are aimed at versatility.

The distribution of NZR wagon types is presented in Table 1.

(3) Wagon Design Concept and Determination of Major Assemblies

Since railway trains need the ability to negotiate sharp curves and are limited in cross-sectional area, any practical train has a degree of articulation incorporated in its design, articulation being structural measures enabling the train to conform to curved track. Articulation serves several purposes: to maximise the usable width of track; to reduce the number of supporting wheelsets in order to reduce mass, aerodynamic drag and vehicle first cost; and finally to assist the

BOGIE WAGONS

CLASS	DESCRIPTION	NUMBER
C	coal hopper	20
Dd	car container	10
E	NZR internal use - special purpose	276
P	box - coal	51
Pk	container	201
Rb	high side	2
Rp	high side newsprint	100
Sp	sheep	3
U	flat top	6
Ua	gas store holder	2
Ub	flat top	344
Uba	air freight cargo	4
Ubc	cement tank	58
Ube	rail air furniture	2
Ubh	aluminium sulphate tank	11
Ubj	flour tank	2
Ubl	lime container	4
Ubm	flat top - motor cars	28
Ubs	sulphuric acid tank	8
Ubw	flat top	16
Uc	oil product tank	252
Uca	tank	30
Uct	tallow tank	16
Ud	well	5
Uda		10
Udk	container well	6
Uk	container	1846
Uka	container low deck	59
Ukc	container coal	4
Ukp	container plaster	4
Ukx	container	9
Ul	flat top logging	71
Un	flat top	100
Ur	flat top	412
Urc	cement tank	31
Url	flat top logging	67
Urt	flat top logging	17
Us	flat top	339
Usa	sulphuric acid tank/CO_2	16
Usd	lime/flour tank	3
Usf	container flour	17
Usg	LPG tank	10
Usk	container	148

Table 1. Population Distribution of NZR Wagon Types (13/4/82)

BOGIE WAGONS (cont'd.)

CLASS	DESCRIPTION	NUMBER
Usl	flat top logging	251
Usp	pulp	1
Uss	caustic soda tank	9
Ut	bulk molasses tank	1
Vb	box insulated	12
Vr	box insulated	78
Vs	box insulated	24
Z	box	256
Za	box	424
Zc	box - palletised cans	1
Zm	box - newsprint	75
Zp	box	498

FOUR WHEEL WAGONS

Ba	bulk stock food	2
Bc	containers - bulk powder	123
Bf	bulk flour	9
Bfk	bulk flour	11
Bgk	bulk liquid	10
Bkc		6
Bp	pitch	1
Bt	bulk tallow tank	4
Btk	bulk tallow tank	28
Bxc	bulk molasses	1
Cp	concrete pipes	1
E	NZR internal use - special purpose	386
E others	NZR internal use - special purpose	82
H	cattle	20
Hc	cattle	19
J	sheep	4
Jc	sheep	20
Kc	box	11
Kp	box	2488
Ks	box	1458
Ksa	box	2
Ksc	box	11
Ksp	box	2
Ksx	box	2
La	high side	4253
Lb	high side	1269
Lbf	high side - bulk fertiliser	218
Lc	high side	6043
Lp	high side - newsprint	390
Lpa	high side - newsprint	597
Lpx	high side - newsprint	3

Table 1. Population Distribution of NZR Wagon Types (13/4/82) (cont'd.)

BOGIE WAGONS (cont'd.)

CLASS	DESCRIPTION	NUMBER
Lw	high side - woodchip	12
M	low side	3
Mc	low side	271
Mcc	low side - motor car	101
N	runner	1
Na	flat top	444
Nb	flat top	170
Nc	flat top	786
Nd	flat top	12
Ne	flat top	100
Nf	flat top	299
Nh	flat top	48
Np	flat top	1
Nr	runner	4
Ns	flat top	20
Nx	flat top	1
Q	coal hopper	480
W		40
Xc	box - ventilated	32
Xp	box - ventilated	138
Yb	ballast hopper	71
Yc	ballast hopper	198
Yf	ballast hopper	25
Yh	ballast hopper	85

Table 1. Population Distribution of NZR Wagon Types (13/4/82) (cont'd.)

vehicle in guidance through curves. A range of concepts are applicable, but most commonly applied are the detachable elastically coupled vehicle with two multi-axled bogies (the bogie wagon) and the detachable two axled elastically coupled vehicle (the four wheeled wagon). Wagons have been built semi-permanently coupled with a rotational joint in the body structure, the joint supported by a wheelset suspension or a bogie suspension.

Some of the factors that influence the wagon design concept include:

- size and weight of the lading;
- existing loading/unloading land based facilities;
- the first cost of the suspension to meet the required performance criteria;
- weight and first cost savings (for example, in going to a bogie wagon from a four wheel wagon savings per unit of payload in brakegear, loading/discharge equipment, draw gear can be achieved);
- the amount of payload space idle when the vehicle suffers a breakdown or is under repair;
- reduction in coupling slack;
- height restrictions which can be met by an increased number of smaller wheels;
- frequency of vehicle uncoupling;
- maintenance costs.

(4) Preliminary Design of Major Assemblies

Almost all wagons consist of the following generic assemblies:

- wagon structure;

- draw gear;
- suspension;
- brakes;
- wheelset.

Brakes, draw gear, suspension and wheelsets have been of comparatively limited variety in NZR due to standardisation and the need for compatibility. These assemblies have often been selected from the range of existing designs in the early stages of the design process whereas the structural bodies have been designed to suit the lading and handling requirements.

The dynamic space envelope (the space the structure can occupy without interference) evolves during preliminary design as the arrangements of doors, equipment, suspension, and so on, become firm. For example, on container wagons, the width, length and deck height are chosen to suit container locating and securing equipment and the load gauge. The deck height can be minimised for maximum utilisation of load gauge (this is a requirement for high cubic capacity wagons) but consideration must be given to the height of loading platforms, the dynamic wheel and suspension clearance envelope, draw gear interference with deck and/or lading. Sufficient depth of the structure (for strength) must also exist at the suspension reaction points, the height of which may be known if an existing suspension design is to be used. The location in the longitudinal direction of the suspension reaction points must allow horizontal and vertical curve negotiation. The location of these points are also influenced by their effect on the distribution of bending moments and the like.

(5) Materials Selection

Selection of the body structure material(s) is an important activity in the preliminary design stages. Higher strength to weight ratio materials improve payload per gross tonne but may involve a larger capital expense. Other factors must also be considered such as:

- corrosion resistance (with regard to the commodity) and the effect of abrasive wear on corrosion resistance;
- availability of material;
- coefficient of friction and its influence on efficient unloading of bulk goods;
- existing workshop practices and experience with the material;
- susceptibility to wear;
- utilisation (distance travelled per year) of wagons as a factor in the economics of the service;
- construction costs;
- dynamic force resistance;
- ease and cost of repair;
- cost of hauling a unit of mass for a unit of distance;
- capital cost of locomotives required.

Materials such as aluminium alloy, mild steel, wood, low alloy steels and composites have been used in wagon construction in the past.

5. DEVELOPMENT AND DETAIL DESIGN

Based on the performance specification developed in the planning phase, the development and detail design phase develops the preliminary design to the stage when drawings, specifications and instructions are released for production of the design. Major emphasis is placed on

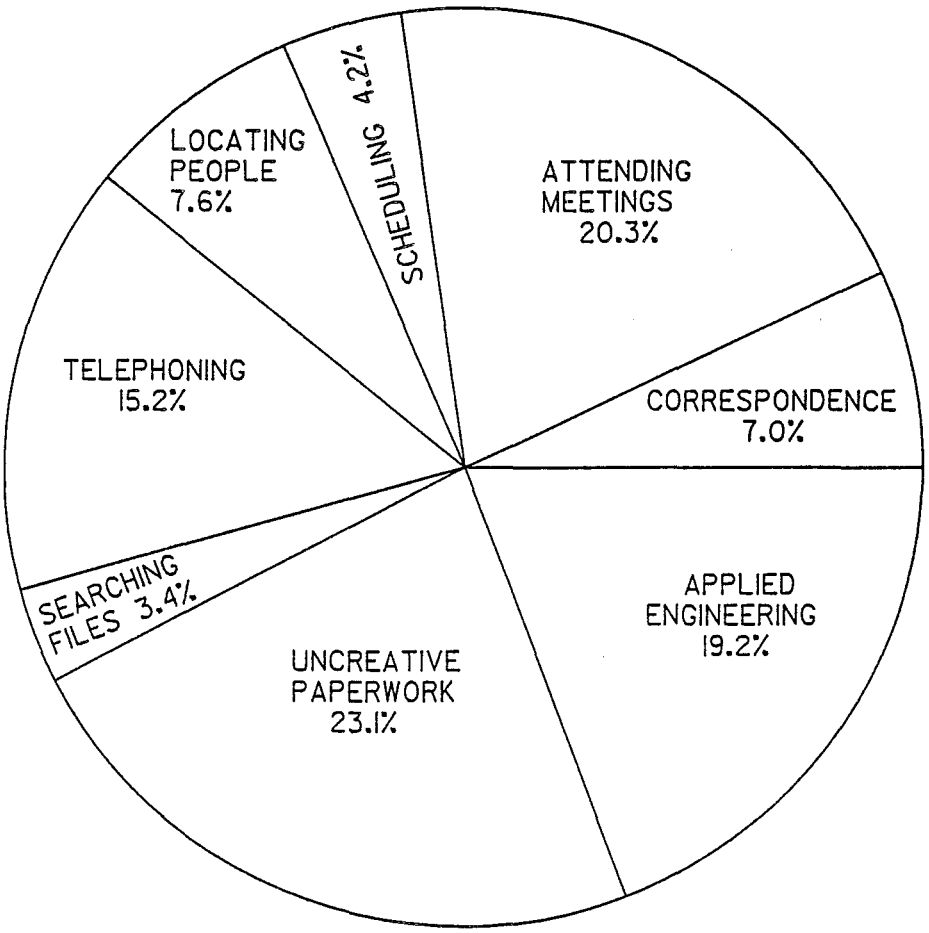


Figure 3. Average Amount of Time Spent on Activities by Engineers (Source:Liker and Hancock,1985)

demonstration of the technical soundness of the wagon and wagon components. The development and detail design process is discussed in more detail in the next five chapters.

(1) The Range of Activities in the Wagon Design System

Figure 3 presents the activity profile for engineers from an American study (Liker and Hancock, 1985). A profile for NZR wagon designers would also probably show a high percentage of activities similar to those performed by officer workers in service industries, as compared to the design and analysis activities studied in this thesis.

Technical office automation products such as document composers, project management tools, spreadsheets, electronic mail and personal time management tools apply computers to the activities other than applied engineering.

(2) Interaction with Wagon Design System Environment

Running concurrent with concept studies, detail design and production planning, may be testing and development activities which will provide information to help resolve outstanding doubts and/or indicate the necessity of design changes. Design reviews with Production Section, Stores, District Mechanical Engineers and Workshop staff may be held throughout these stages to ascertain the availability of materials and proprietary parts to review whether the parts can be made and assembled as planned, can it be serviced, how long will it last, what will it cost to maintain, what it is costing, how long it is actually taking, and to agree to improvements and to record these design changes.

(3) Output of the Wagon Design System

Using information contained in the concept study the Design Office is expected to produce manufacturing specifications or tender specifications depending on the business plan. If a "design and build" tender specification is to be produced (as in the case of the Uca wagon) the tender documents include a main arrangement concept drawing, material and design standards to be adhered to, constraints imposed by NZR infrastructure (for example, operating speeds, curves, loading gauge, draw gear and brakegear compatibility, and so on) and perhaps, in addition, specify certain design features such as the design of bogie suspension. If the specification is for NZR manufacture or contracted construction it will include working drawings with a list of parts containing part numbers and descriptions (ordering description of a proprietary part) and quantity per assembly as well as written instructions.

A Loco-204 form is normally used to initiate all alterations and builds of wagon stock within NZR. These authorisation forms require the Chief Mechanical Engineer's (CME) approval, Account Numbers from the accounts section, drawings and specifications from the Design Office, while the Production Section supplies priorities and addressee of the instructions.

The Production Section and Workshops use the drawings for activities such as jig and tool design, material lists, cost estimates and work planning. The Design Office may also have to produce semi-pictorial diagrams and pictorial views in order to present information in an appropriate manner to customers, wagons inspectors, manufacturers, maintainers, and operators.

(4) Classification of Designs

Classification systems assist retrieval of designs by systematic grouping of like objects according to shape and form features, function, manufacturing processes, and so on. A classification system that enables a wagon designer to retrieve parts with specified shape features will, when a component already exists that meets the criteria, avoid new component design.

An earlier system, OPITZ, was applied to the Uk wagon (295 components) in order to provide data for the design or selection of a classification system. The OPITZ system is based on machine shop components where 60% of the components are rotational. Figure 4 shows that for railway wagons this is not so, rather 65% are non-rotational components. A classification system for wagon design will need to take into account the predominance of long and flat non-rotational components. Over 90% of the components in the Uk wagon were of mild steel, the remainder made up from small numbers of case or surface hardened steel, non-ferrous metal and alloy steel components. A large group (60%) of components had a maximum diameter or edge length between 160 mm and 480 mm. Components made from plate or sheet predominated (48.2%), and significant numbers were made from bar (22.6%), rolled sections (12.2%) and cast (9.9%).

Other considerations (for example, polycode or monocode) in the design/selection of classification systems are available in literature on the subject (see for example, Burbidge, 1975; Arn, 1975). There are a number of commercial computer-based systems available, which automate aspects of the coding and retrieval process.

Because, even with the most disciplined use of component names, most names reflect the application or function of the part and not the

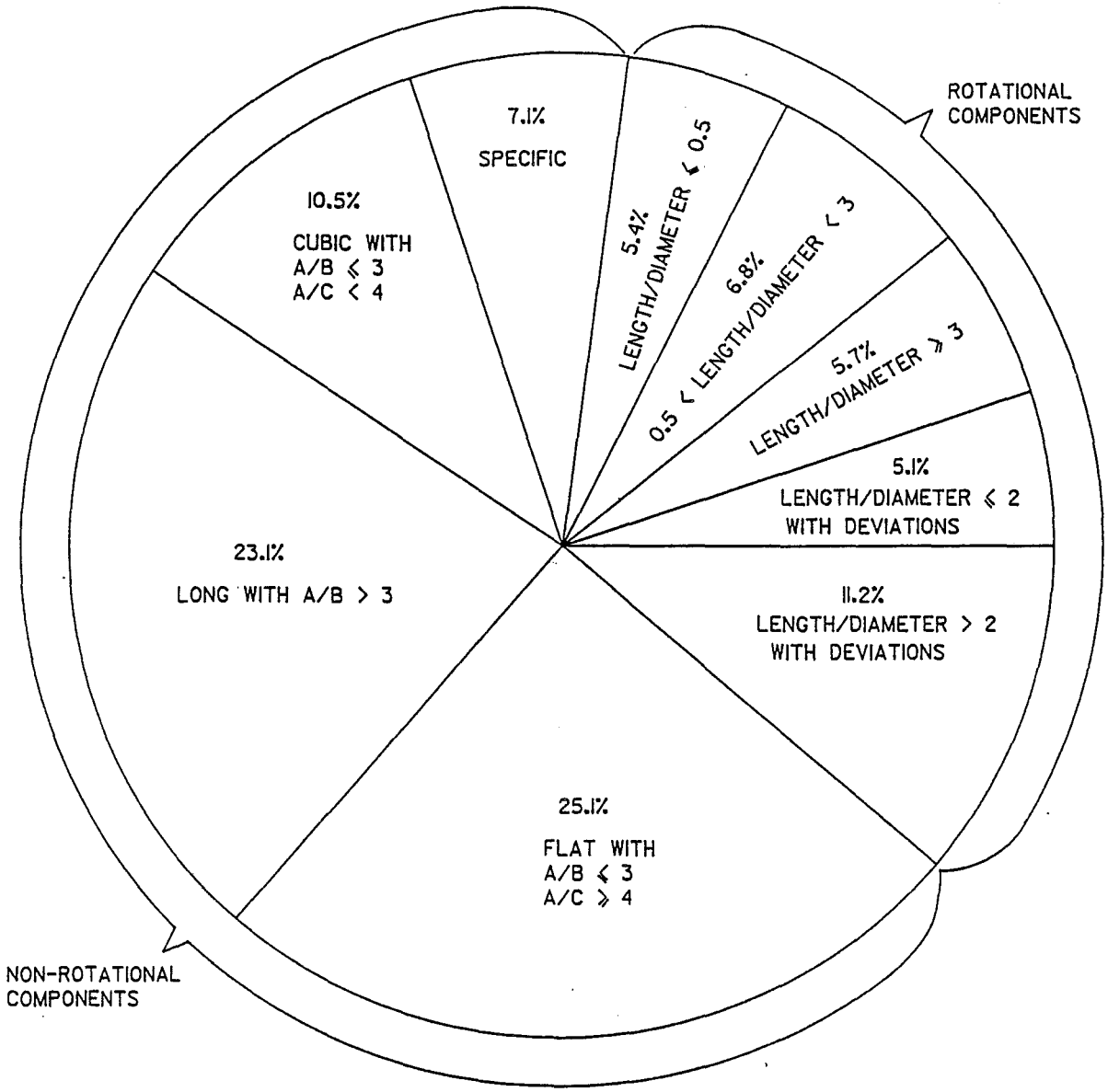


Figure 4. OPITZ Main Geometry Code Classification of the Uk Wagon Components

physical characteristics of the component, classification based on shape is better for variety reduction. It is common to find similar items with different names and different items with the same name. However for more complex assemblies and for some components, classification by function can be performed without ambiguity.

Classification and coding is also used to find part families. Often when designing a part, existing designs are used as a starting reference (consciously or not). A consequence of this is parts that bear some family resemblance but have different dimensions and features in order to meet functional requirements or the preferences of the designer. (Examples of these families are presented in later chapters.) Arn (1975) described a methodology (also used in this thesis) that enables design families to be investigated to determine common basic shapes, shape elements (such as grooves and threads), arrangement of elements and main dimensions. Catalogues of recurring parts, similarity types, and standard parts and computer-based parameterised geometry were produced from this analysis. Recurring parts were defined as the first level in the rationalisation process. Recurring parts were a selection of representative designs from a range of similar parts to be used to reduce variety. Similarity types had varying amounts of commonality (for example, standard basic shape and elements or just basic shape) based on the analysis of frequency distribution of basic shape, and so on, in existing parts. Recommended arrangement of elements and dimension ranges were also included. The parts were initially grouped by part function, and part shape and material classification code. Similarity types offered more design freedom than standard parts but reduced variety and assisted in the formation of standards. Programs that accept designers' input for, or

select on the basis of technical data (for example, allowable stress and load), the size, type and arrangement of shape elements and size of basic shape offer improved productivity and standardisation. (Arn does not cover parameterising shape element size in detail.) Standard parts were those with standard main dimensions as well as standard basic shape, standard arrangement of elements, and standard elements.

6. MONITORING WAGON PERFORMANCE

The present state of knowledge about wagons in their environment is such that testing and development are essential. Communications (for the most part informal and of an anecdotal nature) regarding wagon performance from the Traffic Branch operating staff, Mechanical Branch production and maintenance staff and Railways management inform the wagon design system as to how the wagon designs perform in reality. This information is used in new and remedial design work.

(1) Acceptance Testing in NZR

Acceptance testing involves evaluation of actual functional performance against designed performance. These acceptance tests are carried out at various stages of wear, with a range of representative payloads, speeds, track conditions, and so on, and are usually over a short period of time with respect to component life. The wagon may be subject to ride testing (which involves measurements of displacements, and accelerations), stopping distance trials, shunting impacts, twisted track static tests, and strain gauging under static and inservice loads. Modifications may be carried out as a result of these tests. Tests may also be carried out on individual components or assemblies in

order to determine their performance. The tests are usually carried out only if the wagon or component type has had limited or no service experience.

(2) Component Failure and Maintenance Reporting

Apart from informal feedback, there is within NZR formalised reporting on such things as defective axles (Loco. 96 form), partly and fully fractured tyres (Loco. 90 form), hot axle boxes and derailments (Loco. 69A form and Mainline Derailment report). (A derailment is an incident in which one or more wheels of any vehicle leave the rail or in which the wheels of any vehicle take two different tracks.) The Loco. 64 register is used to record vehicle overhauls, repairs, alterations, paints, air brake overhauls and tares weights. The correspondence pertaining to the derailment of wagon Lb8336 on 30/1/80 illustrates how this information has been used. In summary, the correspondence showed the derailment was attributed to a missing main spring, but as there was no previous history of main springs moving longitudinally, no design modifications were deemed necessary. (End locating brackets were at the time fitted to all recent four wheel wagons but not to Lb wagons.)

Another feedback channel is the General Manager's Suggestions and Inventions Committee which gives monetary rewards to employees for suggestions where appropriate.

The computer-based Maintenance Information System (MIS) currently being implemented aims to provide maintenance records that can be readily analysed. The existing system was regarded as not providing information that was easily accessible, consistent, up to date and sufficiently detailed. Although MIS is primarily intended for management of maintenance, comprehensive and accurate defect reports,

derailment histories, repair histories and costs, and wagon usage (net tonne meterage) records that are easily accessed will be of significant benefit in evaluating component performance in order to determine whether redesign is required and/or the performance specification needs modification. But as mentioned above in section 3 (2) the interaction between reliability and maintainability must be balanced to achieve minimum whole life cost and optimal availability.

The information MIS will provide on the reliability status of existing products could be used in the specification of reliability objectives during design. These objectives may be in terms of life (mileage) expectancies to certain failure percentages; average failure rate or meantime to failure; or permissible repair rates at specified mileages. Tuve and Thomas (1974) have demonstrated the methods of reliability analysis for freight wagon components. Further details on methods in reliability prediction and verification are available in texts on the subject, for example, Smith (1976). These methods generally require identification of failure modes and analysis of the assemblies to provide reliability requirements for the parts and components.

The maintenance staff (the depot, workshop and district engineering staff) can often make valuable input to the design process. Lye (1981) used the history of four wheel highsider type wagons in NZR to illustrate the breakdown of informal and formal communication between the maintenance and design groups. The earlier design of a underframe member suffered from a corrosion problem, it was changed in the next highsider design, but reverted to a design of similar form to the flawed component in the latest wagon design. He suggested that design proposals be forwarded in the appropriate form to the maintenance staff.

7. PLANNING AND CONTROLLING ACTIVITIES IN THE WAGON DESIGN SYSTEM

The development and management function in the wagon design system acts to maintain or improve the system's performance.

In NZR, the Designing Engineer instructs the senior wagon designers on the activities to be performed and receives regular verbal reports on progress. The Designing Engineer also approves any output of the wagon design system before it is communicated to the system's environment. The design project is worked upon by one wagon designer or a small group of designers, and is supervised and directed on a daily basis by senior wagon designers. The senior wagon designers also check designs before submission to the Designing Engineer for approval. Major projects are approved by the CME (refer fig. 5).

Drawing and document administration activities include release and issue of drawing and document copies, recall of copies, maintenance of drawing and document records, search and retrieval, and archiving of drawings and documents.

The performance of the wagon design system can be measured in, for example:

- performance of wagons designed by or whose performance specification was developed by the wagon design system;
- number of specifications or designs per unit of capacity (number of new designs, design modifications, and so on produced per year per person or per dollar invested in equipment);
- time taken to produce specifications or designs per unit of resource input;
- skill development time (days in training, years to develop proficiency levels);

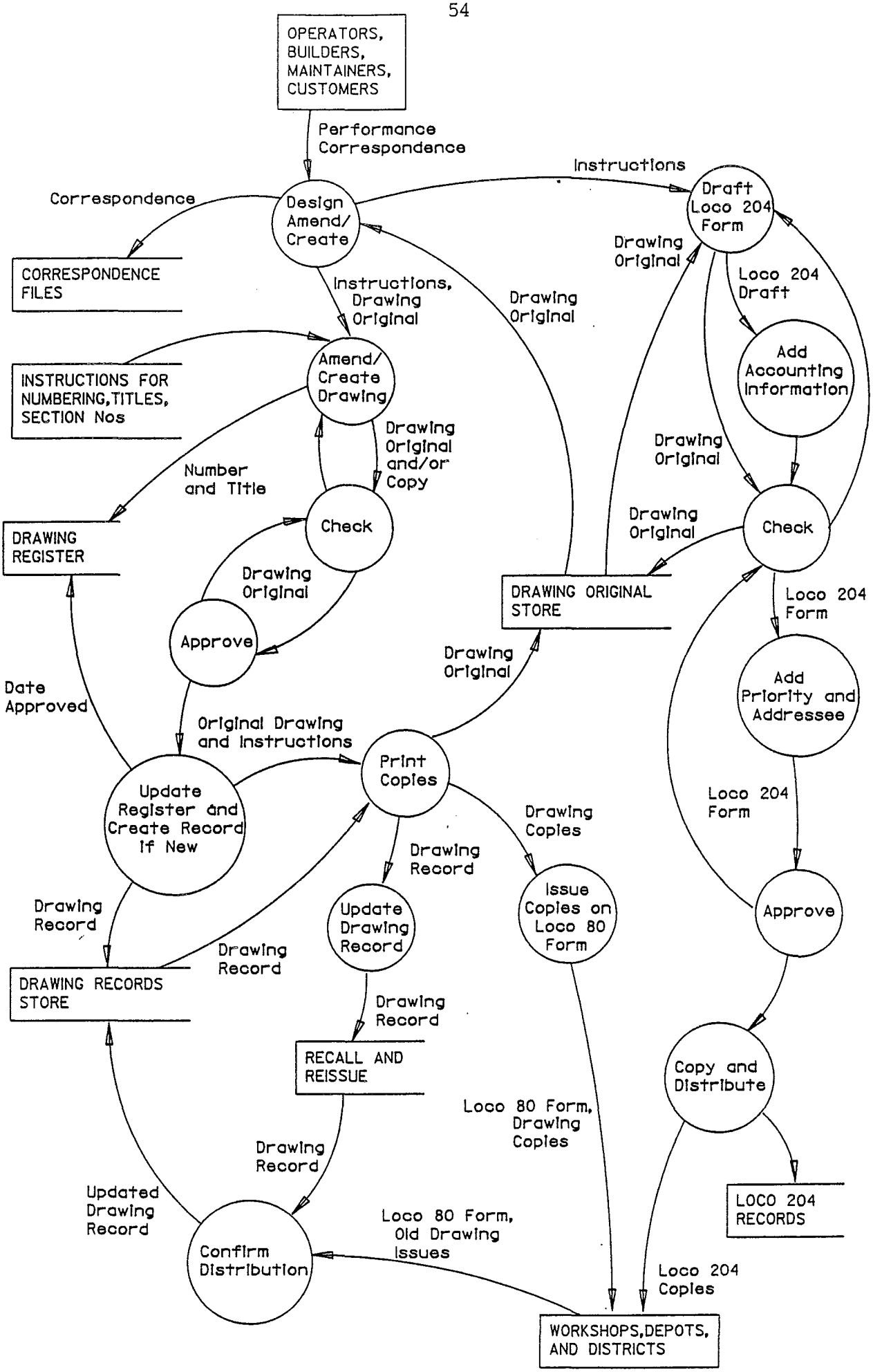


Figure 5. Stages in the Drawing Creation and Amendment Process

- employee satisfaction (measured with questionnaires, interviews, and staff turnover);
- number of designs (drawings, etc) returned for corrections;
- average number of alterations per approved drawing/specification document.

8. REFERENCES

ARN, E.A. (1975) Group Technology : an integrated planning and implementation concept for small and medium batch production. Berlin, Springer-Verlag. 164p.

BURBIDGE, J.L. (1975) The Introduction of Group Technology. London, Heinemann. 267p.

LIKER, J.K. and HANCOCK, W.M. (1985) A Methodology for the Detection of Systems Barriers to Engineering Productivity. Dept. of Industrial and Operations Engineering, University of Michigan.

LYE, H.Y. (1981) A Systems Modification in the Inspection of Wagons at NZR. Christchurch, University of Canterbury. (Project Report : M.E. : Mechanical Engineering).

MUHLENBERG, J.D. (1978) Resistance of a Freight Train to Forward Motion : Vol.I - Methodology and Evaluation. Springfiled, Virginia, NTIS (FRA/ORD-78/04.I, PB-280 969) and Vol. II - Implementation and Assessment. (FRA/ORD-78/04.II, PB 80-118326).

SMITH, C.O. (1976) Introduction to Reliability in Design. New York, McGraw-Hill. 269p.

TUVE, R.F. and THOMAS, R.G. (1974) Reliability Analysis of Freight Car Components Where the Population Is Unknown. ASME Paper No. 74-WA/RT-5.

CHAPTER III

WAGON STRUCTURE DESIGN

1. DEFINITION, FUNCTION, AND PERFORMANCE

The vehicle structure transmits the loads between the other main assemblies and (directly or indirectly) the lading. The major loads are the draft and buff loads applied by the drawgear; the supporting, guidance and braking loads applied through the suspension; the distributed or point loads applied by the lading; and the structure's own inertia.

The problems associated with wagon structures are not new and occur in other large redundant structures. Design involves the specification of a structure that will be safe, and of reliable structural integrity for its required life time (typically 30 to 50 years), and yet be cheap to produce/acquire and cheap to maintain. The ratio of wagon capacity to wagon tare is also an important measure for operational reasons - fuel economy, reduction in track wear, and so on. Although no derailments occurred due to underframe defects in the period 1977-1980 (Vink, 1981), breaking, cracking, and bending of structural components has occurred in the past.

Constraints such as the external vehicle dynamic clearance envelope, the suspension clearance envelope, the equipment installation requirements, door and lading space requirements (perhaps involving inclination of hopper sides or width and height of drums, pallets, etc),

maximum height of centre of gravity (to prevent overturning), and balanced wheel loading, all determine the envelope that can be used by the designer for structural purposes.

2. CONCEPTUAL DESIGN OF THE STRUCTURE

The conceptual design determines the load paths which must lie within the envelope that is defined by the various constraints. Traditionally if the wagon had a superstructure it rode on rigid steel-girder structures known as an underframe (that section of the body structure below the decking) and the relatively fragile superstructure or "body" consisted of framing and sheathing. Internationally the modern structure is generally a stiffened shell with end structures to distribute the drawgear and suspension loads into the body shell, or, in the case of vehicles with no superstructure, a central longitudinal beam structure with transverse structures to distribute lading point loads into the body structure as well as end structures (see fig. 6).

But as discussed earlier, tentative decisions (or at least narrowing down the range of possibilities) on structural layout, material, and capacity have been made by this stage. Other loads on the wagon structure have sometimes been estimated by experienced designers from the information available. Alternatively the loads specified in the various manuals of standards and recommended practices (Assoc. of American Railroads, 1982; Australian and New Zealand Railways, 1972) were used. The tare, mass centroid, and inertias have sometimes been

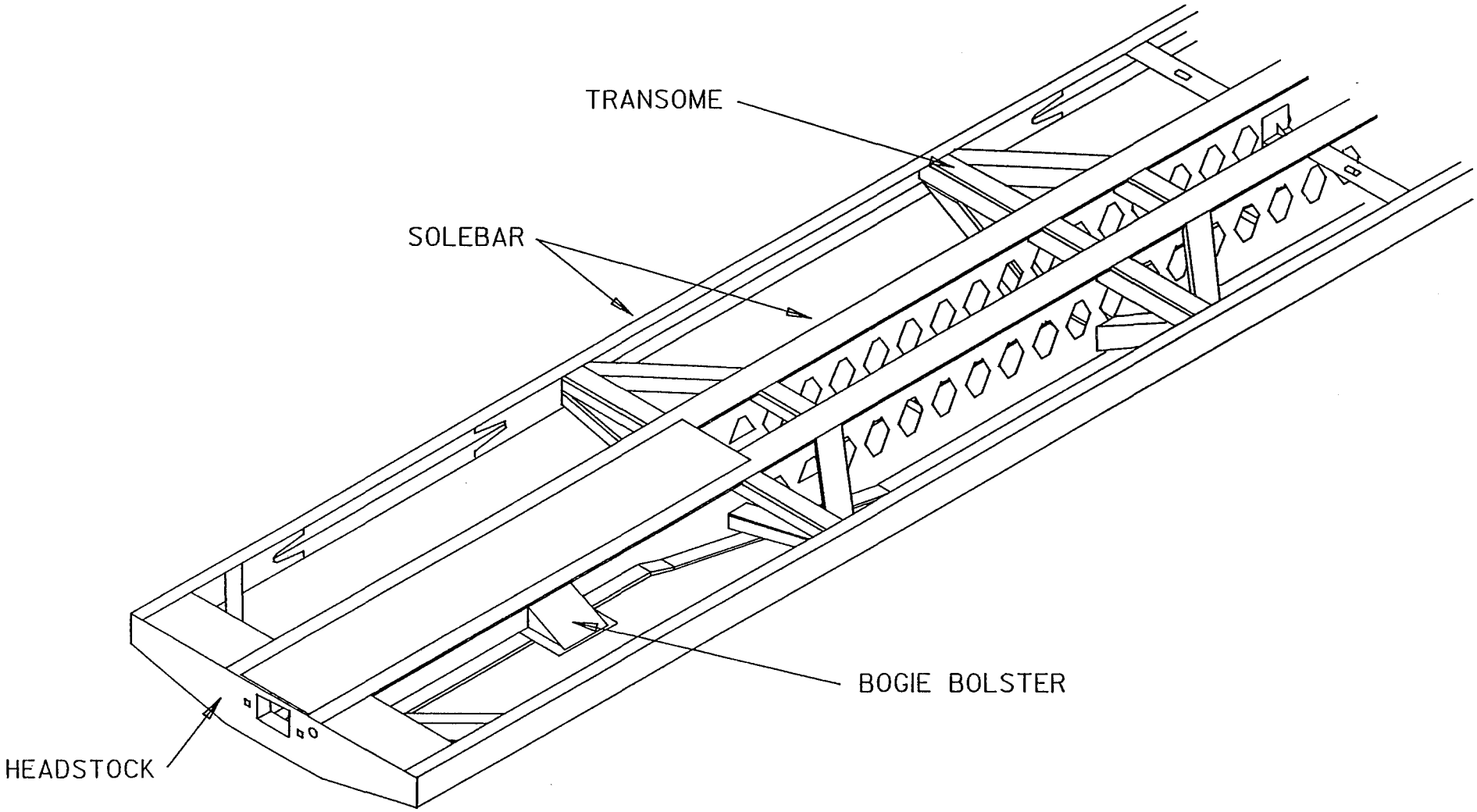


Figure 6. Flat Wagon Structure Arrangement

estimated in the early stages from the weight of existing wagons, and/or wagon assemblies (where these assemblies or ones sufficiently close to them are to be used again) and from approximate major component weights, weld allowances, and their tentative positions. These estimates were revised as the design proceeded and more detail became available (NZR CMEO Design Office : Dd, Pk, Udk, Ucg Design Notes). Estimates of area, inertia, plate thicknesses based on experience or simplified stressing calculations were sometimes made.

3. COMPONENT DESIGN FOR STRENGTH, STABILITY, AND STIFFNESS

The wagon structure now evolves as detail checks are made on its strength, stability, endurance, stiffness, and so on. These analyses are not only performed on new structures but also on existing structures if they are to carry unusual loads, if they are to be modified for a new service, or if inservice structural failures are occurring.

(1) Applied Loads

The analysis of strength and stability usually involves methods typified by those shown in Assoc. of American Railroads (1982), however load magnitudes, distribution of applied loads on the structure, and load path models (how the structure resists the applied loads) may differ from design to design (NZR CMEO Design Office: Uk, Pk, Zm, Ucg, Ks, Dd, Udk, Nk, Nx Design Notes). The components analysed and the failure modes checked also vary between designs.

Whereas in the design notes loads such as the point loads due to container weights were straightforward, those resulting from longitudinal, vertical or lateral accelerations, and inservice loads such as loading and unloading impacts on the floor and sides, coupling impacts, buff (an in-train compressive load on the drawgear) and draft (an in-train tensile load on the drawgear) loads on the roof and ends, and general service lading load, required judgement as to their magnitude and distribution. For although the manuals of recommended practices (Assoc. of American Railroads, 1982; Australia and New Zealand, 1972) specify loads and load combinations, experience and testing was required to determine those representative of local conditions. A load factor to account for the dynamic loading has sometimes been applied.

The body self weight (the tare less the weight of the suspension and wheelsets) has been distributed in a number of ways, for example, it has been added to the lading weight, or apportioned to the major structural components and treated as a uniformly distributed load (UDL) on these members. Vehicles in general service have had a number of loads applied, for example, a central short span UDL, a UDL between the bogie centres, and a full length UDL. The horizontal and vertical loads were combined to determine the worst cases for the structure.

(2) Solebar Design

The solebars are structural components that are frequently analysed to ensure maximum stresses conform to limit criteria. A solebar is a longitudinal beam which carries a major portion of the drawgear and vertical loads, traditionally of standard rolled channel but more

recently other standard rolled sections and fabricated members have been used (see fig. 6).

Bending moments have been distributed to the solebars alone, to the entire cross section (including floor plate, and so on) and to cross-sections in between these extremes. The moments have been distributed amongst the members on the basis of : equal deflection (that is, the moment or load proportioned according to stiffness); the moment distribution method; treating the cross-section as an integral beam; and a design factor (varying from equal distribution amongst all solebars to equal distribution on only two of the members). The decision on which method was used appears to depend on the applied loads, the structural configuration, and the designer who performed the analysis.

End loads have usually been applied to the central members before the bolster (a transverse beam in the underframe over a bogie which transmits the load carried by the longitudinal members to the bogie through the centre plate) or in the case of two axled wagons, the first major transome behind the drawgear. On the section of the underframe between the bolsters or major transomes, the end loads have been distributed to a greater number of members as judged appropriate by the designer (NZR CMEO Design Office : Udk, Dd, Ukx, Pk, Zm, Ucg, Nk, Nx Design Notes).

The solebars may have curved lengths as in well wagons, and/or tapered sections, and/or may be part of a trussed underframe. These geometric property variations change the eccentricity of end loads and consequently must be allowed for in the underframe analysis. The solebars have most often been considered as simply supported at the suspension connections.

Although a considerable number of loads and combinations of loads may be applicable and many locations may be critically stressed, the designer has usually investigated buff, draft, tare, and some lading loads singly and in combination at selected positions on the solebar (for example, where changes in section geometry occur, at the centreline of the wagon, or position of maximum bending moment, and where point loads have been applied).

The stress limit which, although specified in the manuals of standards and recommended practices, has varied somewhat in the design notes. As well as acting as a constraint on calculated stresses, it has also been used to determine the required section area and modulus. Formulas and rules in Assoc. of American Railroads (1982), BS449, and other sources have been used to compute shear stresses, equivalent stresses, shear resistance of webs, critical local buckling stresses, combined bending and axial stress limits and maximum unsupported lengths of solebars (NZR CMEO Design Office :Pk, Udk, Dd Design Notes).

(3) Transome Design

Transomes have often been analysed to ensure they meet the design stresses based on yield and buckling criteria. They have been of truss construction (such as the Uk intermediate transome), rolled shape beam construction, and fabricated beam construction (for example, bogie transomes and Nk link suspension transome). Again the method of analysis varied as judged appropriate by the designer considering the structural configuration and the nature of the loading, for example, jacking loads, fork lift truck loads, lading point loads and suspension reactions (NZR CMEO Design Office : Nk, Dd, Udk, Ucg, Zm Design Notes).

The solebars, spacers and transomes have their ends shaped to match the adjoining component and may have drilled holes for, say, the bogie check chain or draft lugs.

(4) Other Components

Other principal load carrying members which have been analysed less frequently than solebars and transomes with similar methods include floor plates, shear plates, floor stringers, headstocks (transomes at each end of the underframe), doors, roof and ends (NZR CMEO Design Office : Ucg, Zm, Ks Design Notes).

Frequently shear force and bending moment diagrams have been prepared as well as diagrams of the loaded structure. Summaries tabulating the critical load condition, the position on the member, the stress and a name or size identifying the design trial have also been prepared.

Some assemblies and components have been used repeatedly, for example, the bogie centre assembly (one of a pair of cast plates with associated components which fit into the other and support the wagon body on the bogies allowing them rotational freedom) and float blocks (blocks that carry part of the weight when the wagon is tilted) (see NZR CMEO Dd, Pk, Uk, Za, Zm, C, Us, Rp, Udk, Usg, Uda drawings). Other assemblies have been formed from common components. For example, headstocks in the Pk, Zm, Za, Upa designs are constructed from the same die pressing, side chain angle, drawbar rest plate, and similar strengthening plate and gussets, but have varying positions and sizes of the holes and cutouts (see fig. 7). Other wagon designs have headstocks of similar form, as are many of the bogie transomes. Roof,

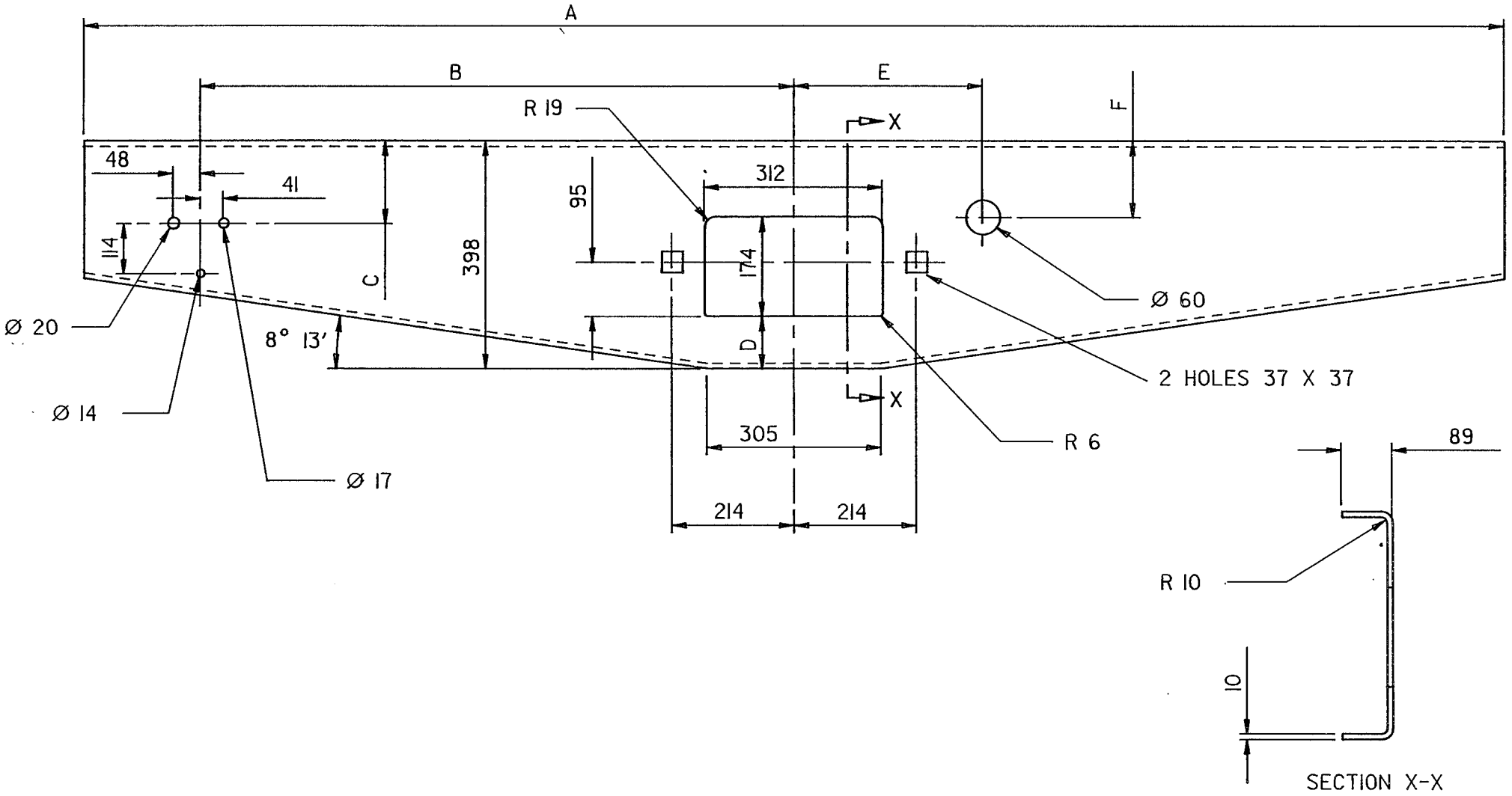


Figure 7. Zm Headstock Parameterised Geometry

door and end die pressings have also been used repeatedly in different wagons.

Campbell and Jenks (1961) have presented a case study of the design, and in particular, structural analysis and testing of a covered hopper car, the analysis was similar to that used in the design of Design Office's C wagons (NZR CMEO Design Office C Design Notes). More recently, Milton (1984) has reported on the structural analysis of an existing and a new hopper wagon in which use of the finite element method was made to obtain a better understanding of load paths, to determine inservice loads that would result in side sheet buckling and to acquire stress ranges for fatigue assessment.

(5) Design for Deflection

Excessive deflection of the body structure can result in brake gear stretch, brake pipe stretch, eccentricity of end loads, and interference with track structures. However these effects can be minimised by positive camber. For slender wagons (large wagon length to depth ratio) large deflections may indicate low natural frequencies so increased stiffness may be required and for some wagons it may be the limiting criteria. The limiting deflection has usually been based on experience and checked against deflections calculated using for example, McCaulay's method, moment area method, simple beam formulas, or the finite element method (NZR CMEO Design Office : Udk, Dd, Uk, Zm, Nx Design Notes).

(6) Computers in Structural Design for Strength, Stability, and Stiffness

The Design Office has performed finite element analyses using a

package on the State Services Commission Vogel Centre computer. Space frame idealisations have been used to check the internal member loads and stresses in manually configured and sized structures and to gain a clearer appreciation of load sharing mechanisms (NZR CMEO Design Office :Pk, Usl, Btk, Zm, Ks Design Notes). They have also had a stress analysis using beam and plate elements performed on a tank wagon (Ucg). Brackets, members and other details that were judged to have little effect on the structural stiffness were omitted from the analyses, possibly to be stressed later. However, not all finite element analyses have been tailored to the understanding of the overall vehicle structure, as many published reports deal with components and local stress concentrations (see, for example, papers in Moyar, and others 1978).

Pre- and postprocessors, that is, programs concerned with getting the finite element model onto the computer and getting the response out, can enable rapid discretisation of a rail vehicle structure and facilitate detection of errors significantly reducing analysis time (Steven and others, 1981). Most pre- and postprocessors developed rely heavily on the use of interactive graphics.

Use of computers in wagon structural analyses has not been restricted to applications of the finite element method.

The Design Office have produced a program for their HP9825 that iteratively solves for the distribution of loads in a trussed underframe by treating the solebar as a continuous beam on elastic supports. While Hunt (1983) produced a set of interactive programs for checking member sizes with procedures specified in Assoc. of American Railroads (1982). Both the relationship between the design variables and the

decision tables representing the decision processes were presented to the system and used by it as data. Therefore revision or addition to the design procedure could have been done without reprogramming except where evaluation of new logical or numerical expressions were required. The conditional execution of the design procedure involved the computer program deciding the sequence of generating data in the light of the data available at the time of execution. The entire set of design variables were repeated several times in the data storage files in order to allow sets of standard data and case studies, but the scheme maintained consistency within a particular case study if some input values were changed by setting a status flag for the affected variables. Programs to produce and maintain the design procedures as well as to design wagons were developed.

Whatever method of analysis is used, estimation of the effective section properties is required. Most section properties for proprietary shapes, plates and bars, can be obtained from catalogues and standards, but section properties of fabricated sections, castings, forgings, and so on, require calculation. Computer programs have been developed for estimating section properties. For example, BEAMSTRESS described in Pilkey and Thasanatorn (1978) determines comprehensive section properties and stresses in an arbitrary homogeneous or composite cross section of a straight bar (see also Garg and others, 1977).

(7) Joints

Welds are the predominant type of structural connection although other fasteners have been used, for example, rivets, huck bolts and threaded bolts, for reasons such as welds had unsuitable mechanical

properties or a removable assembly was required. Not all connections have been "stressed", the designer judged which were worth investigating on the basis of the type of weld, weld length, and so forth. The various design codes specify such things as minimum and maximum hole pitch, minimum and maximum hole to edge distance, allowable stresses, minimum and maximum weld sizes and lengths in order to avoid plate shear and tearing, moisture entry, plate buckling, bolt yielding, edge dishing, inability to tighten bolts with wrenches, and so on. The allowable stresses have been used to check the capacity of the joint against the design load, the design bolt diameter against minimum allowable diameter, and to determine required plate thickness, number of bolts, weld length, and so forth. From weld or bolt group geometry the identification of possible critical points and determination of group properties such as shear area and centroid can be made. The effect on production costs of factors such as the number of joints, bolts or parts, the size of bolts, the number of bolt types, installation effort required, and use of standard sizes and materials appears to have been determined intuitively. Wallace (1983) and Hogan and Thomas (1983) are illustrative of the attempts at computerisation of joint design.

(8) Design for Manufacture

The final design is also influenced by manufacturing considerations such as available stock, ease of assembly and construction, cost of achieving tolerances and positional accuracy required, machining and welding introducing residual stresses and unwanted curvature and deflections. The designer's and manufacturing staff's experience appears to have been the major source of input to the design in this area.

4. DYNAMICS OF THE STRUCTURE

Assessment of the dynamic characteristics of the body structure has sometimes been performed. These assessments were performed in order to avoid undesirable vibration under operating conditions, which would result in fatigue damage to components, damage to lading, and possibly catastrophic derailment. An estimate of the fundamental flexural frequency using a text book beam vibration formula was compared with a rail joint excitation frequency in NZR CMEO Design Office Dd Design Notes.

As well as detuning the body structure to avoid the suspension natural freqencies, primary sources of impulsive excitation (for example, rail joints) and periodic excitation (for example, out of balance wheel rotation) dynamic analyses have also been performed to determine dynamic stresses. Dynamic load factors, based mainly on track testing (see Anon, 1972; and Australia and New Zealand Railways, 1972), have been used as a rule of thumb method for calculating dynamic stress range when designing wagon structures against fatigue. Moyar and others (1978) have reported on efforts to verify flat wagon analysis models with test results. The work in Moyar and others described a range of modelling techniques to obtain time domain response, amplitude and phase angle of response relative to forcing function (frequency domain response or transfer function), as well as natural frequencies and mode shapes. It was suggested simpler beam models could be employed in the initial stages of design because they required less computing

time and preparation manhours, than the more detailed beam and plate finite element models. However the detailed models were more accurate for the higher frequencies and for internal loads. In Assoc. of American Railroads (1978) the fatigue life predicted from time history response of a space beam finite element model was compared with that calculated by proportioning static stress values with a "g-factor" load spectrum.

5. ENDURANCE OF THE STRUCTURE

(1) Fatigue

Assessments of fatigue performance are calculations to ensure the proposed design will achieve a satisfactory life or to estimate the life of an existing structure.

(a) Design for Fatigue within NZR Design Office. In the past the stress ranges have been taken from those calculated manually or by finite element analysis. For values derived by static analyses an appropriate range or dynamic factor was applied by the designer based on service testing or design codes. The designer also determined at which locations to conduct the detailed fatigue analyses usually in the light of previous fatigue failures, the current state of the design, and stress and dynamic analyses. Then these stresses were compared with the allowable values for the class of member or joint as defined in the various codes (for example, BS 153, 449, 2573), or estimates of life were obtained from the standard stress life data using these stresses. The estimated fatigue life (number of loading cycles) was compared, taking into account wagon utilisation, with the estimated number of

loading cycles occurring in the wagon's life. The magnitude and frequency of the fatigue loads were based on the designers knowledge of service conditions sometimes supplemented with information from the freight operating sections of NZR (NZR CMEO Design Office : Ks Zm, Nk, Pk, Dd, Nx design notes).

(b) International Methods of Assessment. The Assoc. of American Railroads (1982) have recently introduced a specification for fatigue analysis of freight wagons and have made available computer programs incorporating these fatigue analysis and data reduction techniques (Zarembski, 1978). The designer selects appropriate acceleration and/or load occurrence spectra, and a modified Goodman diagram containing geometry influenced material properties for the structural detail under consideration (for example, welded connections or beams with welded reinforcements) from those supplied. The spectra were developed from stratified random sampling of vehicle service loads and accelerations, and contain the percentage of total occurrences for each maximum to minimum load cycle. If no suitable modified Goodman diagram is available then the designer must use the basic material properties and multiply the nominal stresses by an appropriate fatigue notch factor. The stress spectrum must be determined from the load spectra by an appropriate method. The service life (that is, till a crack can be detected by the unaided eye) under conditions of variable amplitude loading is then predicted using the linear cumulative damage hypothesis and an idealised S-N curve.

SAFEM (Prasad and Singh, 1980) is a computer package for computing the fatigue life of structures that have been modelled using the general purpose SPAR finite element system. It incorporates the Assoc. of

American Railroads fatigue analysis specification and the fatigue life and cycle counting calculations are based on the programs mentioned above. SPAR and SAFEM are each composed of a number of "processors" (or program modules-19 in SPAR, 3 in SAFEM) that are run individually and communicate with a "data complex" or database which contains information generated or used by other processors. The individual processors can be assembled into a sequence in order to achieve a goal in a particular manner, for example: evaluation of the fatigue life of an existing design, with no intention of redesigning the given design; submission of a batch job to make a fatigue evaluation, review results at a later date and possibly modify the design; design a component based on more than one set of load spectra continuously interacting with the program.

The set of processors contain data management functions such as input and editing of data, transfer of data to a printer or a sequential file, changing data set names, retrieval and graphical display, as well as finite element analysis and fatigue life analysis functions. All user input is in free format, some supplied through question and answer dialogues, the rest in data cards (except for the digitized acceleration/load history which is supplied in a sequential file) which can be used to change default values of control parameters and other data. The output data which is to be retained for communication to other programs is given a "data set" name and sent to a "library" file. A data set is a predefined group of data referred to and accessed as a single entity, each data set being identified by two integers and two strings of four alphanumeric characters. A table of contents exists for each library and is used to describe (in terms of names, disk addresses, size, data format, insertion date, and so on) all of the data

sets resident in that library.

Described in Musiol and others (1981) is the computer- based fatigue analysis package, FATS-11, for use by designers which is being developed by the British Railways Board Research and Development Division. FATS-11 is a suite of interactive programs using graphical displays that enables each stage of the analysis to be assessed. The "critical location" or "local stress-strain" approach is used for suitable problems (see also Watson and Rebbeck, 1975). Otherwise (for example, welded structures and components where fretting is significant) the more traditional fatigue analysis which employs constant amplitude S-N data and design spectrum of stress levels is used to calculate life to crack initiation. Under the control of a general manager module, which provides on-line help facilities and option displays, are the analysis programs and database. The databanks include: the fatigue environment containing turning point histories obtained from service measurements and artificial histories of loads, stresses, strains and deflections; material properties containing chemical composition, heat treatment, monotonic and cyclic stress-strain mechanical properties and so on; stress concentration information; data on welded details for a procedure that can be used as an alternative to one based on BS5400 data; and structural fatigue data including S-N curves. As well as formatted storage, some of the databanks offer keyword and multi-parameter retrieval and display facilities and the load databank also permits editing of existing histories and generation of new histories. Redesign facilities such as allowable stresses to achieve specified life, automatic provision of alternative geometry changes or weld descriptions were provided or were intended to be provided.

With regard to this thesis comments made by Musiol and others in their concluding remarks are worth repeating. "The paper has shown how interactive computer-based methods can be used to make the most advanced techniques of fatigue analysis available to the non-specialist. Emphasis has been placed on the output of interim results, using computer graphics, to enable the user to develop an understanding of the fatigue processes involved.... Meanwhile, research is continuing in each of the areas covered by these programs, and the results, in the form of improved data or more accurate methods of analysis, will be included as they become available."

The AAR have published reports (Pellini, 1978; Pellini, 1979) relating defect size (particularly in welds and castings) to stress concentration factors and S-N curves. The reports contain discussion on economic and technical factors affecting the choice of stress concentration factors due to defects (and therefore defect size and maximum inservice crack size). These factors included criticality of intended service of the structural component or connection (for example, those regions whose failure would result in derailment, train partings or personnel hazard maybe more critical than those whose failure would cause lading contamination or damage), whether the wagon will be in general or specific service conditions, the implications in quality control and production cost, and vehicle weight.

(2) Fracture

When initial defects and allowable inservice cracks are of less than critical size for fast fracture (brittle fracture) or plastic

collapse under service loads, fracture mechanics can be used to study the effects of cracks on structural integrity. Consequently, inspection periods and withdrawal from service criteria for crack structures may be determined on a rational basis. For components that cannot be easily and cheaply inspected, they can be designed with working stresses, maximum allowed built-in defect size and material fracture toughness such that fatigue crack propagation from the built-in defects is either non-existent or insignificant during the expected service life of the component (Cannon and Allen, 1974). After component failure, fracture mechanics can also supply the designer with useful information such as component failure stress.

(3) Corrosion

Corrosion, the deterioration of a material because of chemical reaction with its environment, can seriously affect the life of structural components such as superstructure panels and framing. Although the designer may not be able to effect much change on atmospheric conditions and the like, the designer can influence a component's local environment by avoiding features such as corners, ledges, and grooves that will trap moisture and dirt. Other means of corrosion control include choice of structural material, choice of protective coating or barrier films (which may be employed to control corrosion on an existing structure), and cathodic protection (or avoidance of potentially corrosive metal junctions) (refer Edwards 1957; Gellings 1976, for further information).

6. OPTIMISATION OF THE STRUCTURE

There has been steady development in the field of structural optimisation, but the literature which describes successful applications of structural optimisation in the industrial design environment is rather limited (Bennett and Botkin, 1982).

Davies and Wang (1972) describe a method of structural optimisation based on the combination of fully stressed and linear approximation techniques, which has been applied successfully to several wagon underframes. The most impressive result obtained was a 43% reduction in the weight of a covered wagon underframe. But because of fabrication requirements and difficulties in finding standard rolled sections to fit the optimised design, the reduction in the weight of the underframe in the optimised production design was about 35%. The underframes were modelled as rigidly jointed space frames using beam finite elements and were subject to multiple load cases. The fixed shape members were sized to minimise the weight of a structure of given geometry and topology considering member size, deflection and stress constraints. The stress constraints were based on the recommendations laid down in BS 153 and BS 449. Dynamic frequency constraints were not considered.

7. BODY FITTINGS AND EQUIPMENT

The fittings and ancillary equipment on the wagon body include generally applied components, and components applied to wagons in specific types of service. Generally applied components include waybill assembly, loadplate, lamp brackets, footsteps, grab irons or hand holds,

and horse hooks. Components applied to wagons in specific types of service - include container twistlocks, tank fittings, tarpaulin ridge poles, hopper discharge equipment, log cradles, end board assembly, stanchions, bondchain assembly, lashing hooks and bars.

All fittings and equipment must not foul the track or structures beside the track. It is also desirable to eliminate loose items which can be dislodged while in service or lost due to lack of operator discipline.

For fittings such as lamp brackets, ferry hooks, jacking pads, and waybill clip assemblies, the number and general location was determined from Traffic Branch regulations and operating practices. The designer appeared to give these components their precise location according to where he judges them to fit best.

The labour union representing the workers who use the wagon were usually consulted over, for example, the placement of grab irons and hand grips if placement affected the safety and efficiency of yard work. Many wagon designs have standard designs or components designed for an earlier wagon.

Devices such as bond chains, stanchions, log cradles, and twistlocks are employed to secure the lading and prevent it from shifting under both motion related forces and environmental forces such as wind gusts, and therefore strength must be considered in their design.

For wagons that are to carry ISO containers, the relative location of the twistlocks for each container are laid down. The positioning of the containers on the wagon has been influenced by the use of refrigerator units or loading practices. For example, the practice of

loading containers while on the rail wagon at freezing works resulted in extra decking between end loading containers for at least one wagon design (Uk). Decisions of this nature were usually made in conjunction with the customer, the Traffic Branch and other relevant NZR departments at an earlier stage - "the conceptual design stage".

As well as strength considerations the opening/closing and locking mechanisms must be designed for correct operation which includes deflection and force analyses. Pins, bolts and shafts are common engineering components that have been used in door equipment. There are proprietary doorgears available.

8. REFERENCES

ANON. (1972) Service Loads and Structural Safety Requirements of Railway Vehicles. Rail Engineering International, 2(8) : 381-383.

ASSOC. OF AMERICAN RAILROADS. RESEARCH AND TEST DEPT. (1978) Freight Car Dynamics - Demonstration Test and Analysis volume II : forced vibration, dynamic stress analysis and fatigue life prediction. (AAR Research and Test Dept. Report No. R-322).

ASSOC. OF AMERICAN RAILROADS. MECHANICAL DIVISION. (1982) Manual of Standards and Recommended Practices. Section C - Part II : Specifications for Design, Fabrication and Construction of Freight Cars. Vol.1, Rev. ed.

AUSTRALIA AND NEW ZEALAND RAILWAYS. [1972] Manual of Standards and Recommended Practices.

BENNETT, J.A. and BOTKIN, M.E. (1982) Automated Design for Automotive Structures. Journal of Mechanical Design, Trans. ASME, 104 : 799-805.

BRITISH STANDARDS INSTITUTION. (1969) The Use of Structural Steel in Building. Pt.2 Metric Units. 120p. (BS 449:1969).

BRITISH STANDARDS INSTITUTION. (1972) Steel Girder Bridges. Pt. 3B. Stresses and Pt. 4. Design and Construction. 64 p. (BS 153:1972).

BRITISH STANDARDS INSTITUTION. (1980) Steel, Concrete and Composite Bridges. Pt.10 Code of Practice for Fatigue. 48p (BS 5400:1980).

BRITISH STANDARDS INSTITUTION. (1983) Rules for the Design of Cranes. Pt.1 Specification for Classification, Stress Calculations and Design Criteria for Structures. 68 p. (BS 2573:1983).

CAMPBELL, R.A. and JENKS, I.H. (1961) Design and Testing of a Self-supporting Aluminum Covered Hopper Car. ASME Paper No.61-WA-236.

CANNON, D.F. and ALLEN, R.J. (1974) The Application of Fracture Mechanics to Railway Failures. Railway Engineering Journal, 3(4):6-22.

DAVIES, G. and WANG, H.S. (1972) Minimum Weight Structural Design : the application of linear programming and fully stressed techniques. Proc. of the International Symposium on Computer-aided Design, Conventry, (PPL conference publication No.11).

EDWARDS, A.M. (1957) Corrosion with Some Reference to

Railway Materials. Railway Steel Topics, 4 (1) : 25-33.

GARG, V.K. and others. (1977) Freight Car Dynamics - Demonstration Test and Analysis : vol.1 - Free vibration study. Chicago, Illinois, AAR Technical Division. 192p. (AAR Research and Test Dept. Report No. R-280).

GELLINGS, P.J. (1976) Introduction to Corrosion Prevention and Control for Engineers. Delft, Delft University Press. 138 p.

HOGAN, T.J. and THOMAS, I.R. (1983) A Computer Program for the Design of Fillet Welds to AS 1250-1981. 31st Annual Conference of the Australian Welding Institute, Sydney. p.49-56.

HUNT, A.A. (1983) Computer Aided Design of the Body Structure of Bogie Railway Wagons. Christchurch, University of Canterbury. (Final year project report : B.E. : Mechanical Engineering).

MILTON, P.F. (1984) The Structural Analysis and Proposed Testing of the State Rail Authority of N.S.W. 100 Tonne Coal Hopper Wagons Designed for Service in Coal Trains. Mechanical Engineering Transactions, I.E. Aust., ME9:4:301-307.

MOYAR, G.J.. and others (1978) Track/train Dynamics and Design: advanced techniques. New York, Pergamon Press Inc. 469 p.

MUSIOL, C.A and others. (1981) Advances in Computer Aided Design Against Fatigue. In Engineering Research and Design Bridging the Gap. p. 67-78. (Institution of Mechanical Engineers Conference publication 1981-7).

NZR. CMEO Design Office. Btk Wagon Design Notes. (Tomsett,

S.E.) 1982.

NZR. CMEO Design Office. C Wagon Design Notes. (McDonald, I.) 1968.

NZR. CMEO Design Office. Dd Wagon Design Notes. (Gowan, P.J.) 1974.

NZR. CMEO Design Office. Ks Wagon Strengthening Design Notes.

NZR. CMEO Design Office. Nk Wagon Design Notes.[1980].

NZR. CMEO Design Office. Nx Wagon Design Notes. (May, D.J.) [1976].

NZR. CMEO Design Office. Pk Wagon Design Notes. [1980].

NZR. CMEO Design Office. Ucg Wagon Design Notes.[1980].

NZR. CMEO Design Office. Udk Wagon Design Notes. (Cassidy, W.G.) 1969.

NZR. CMEO Design Office. Uk Wagon Design Notes. (Dyer, P.) [1969].

NZR. CMEO Design Office. Ukx Wagon Design Notes.

NZR. CMEO Design Office. Usl Wagon Design Notes.

NZR. CMEO Design Office. Zm Wagon Design Notes. [1974].

PELLINI, W.S. (1978) Engineering Considerations in Design Against Fatigue Failure. Chicago, Illinois, AAR Technical Division. (AAR Research and Test Dept. Report No. R-306).

PELLINI, W.S. (1979) Introduction to AAR Guidelines for Establishing Defect-quality Criteria in Fatigue Design. Chicago, Illinois, AAR Technical Division. 28p. (AAR Research and Test Dept. Report No. R-387).

PILKEY, W.D. and THASANATORN, C. (1978) A Simple Beam

Analysis of the Flat Car Structure. In Moyar, G.J. and others. Track/train Dynamics and Design: advanced techniques. New York, Pergamon Press Inc. p. 183-201.

PRASAD, B. and SINGH, S.P. (1980) Software Series : SAFEM : A finite-element fatigue life design program for freight car structures and components. Chicago, Illinois, AAR Technical Division. 158p. (AAR Research and Test Dept. Report No. R-434).

STEVEN, G.P. and others. (1981) F.E.M. on Rails - Experience with Finite Element Analysis of Railway Vehicles. Proc. Symposium on Stress Analysis for Mechanical Design, Sydney. (National Conference publication, I.E. Aust., No. 81/4).

WALLACE, B.J. (1983) Computer-aided Design of Bolted Connections. Christchurch, University of Canterbury. 61p. (Final year project report : B.E.: Mechanical Engineering).

WATSON, P. and REBBECK, R.G. (1975) Modern Methods of Fatigue Assessment. Railway Engineering Journal, 4(6) : 10-20.

VINK, P. (1981) Derailment Statistics for Years 1979 and 1980. NZR Way and Works Branch Research Report 192.

ZAREMBSKI, A.M. (1978) Freight Car Environment Characterization for Fatigue Life Analysis. In Moyar, G.J. and others. Track/train Dynamics and Design : advanced techniques. New York, Pergamon Press Inc. p.205-221.

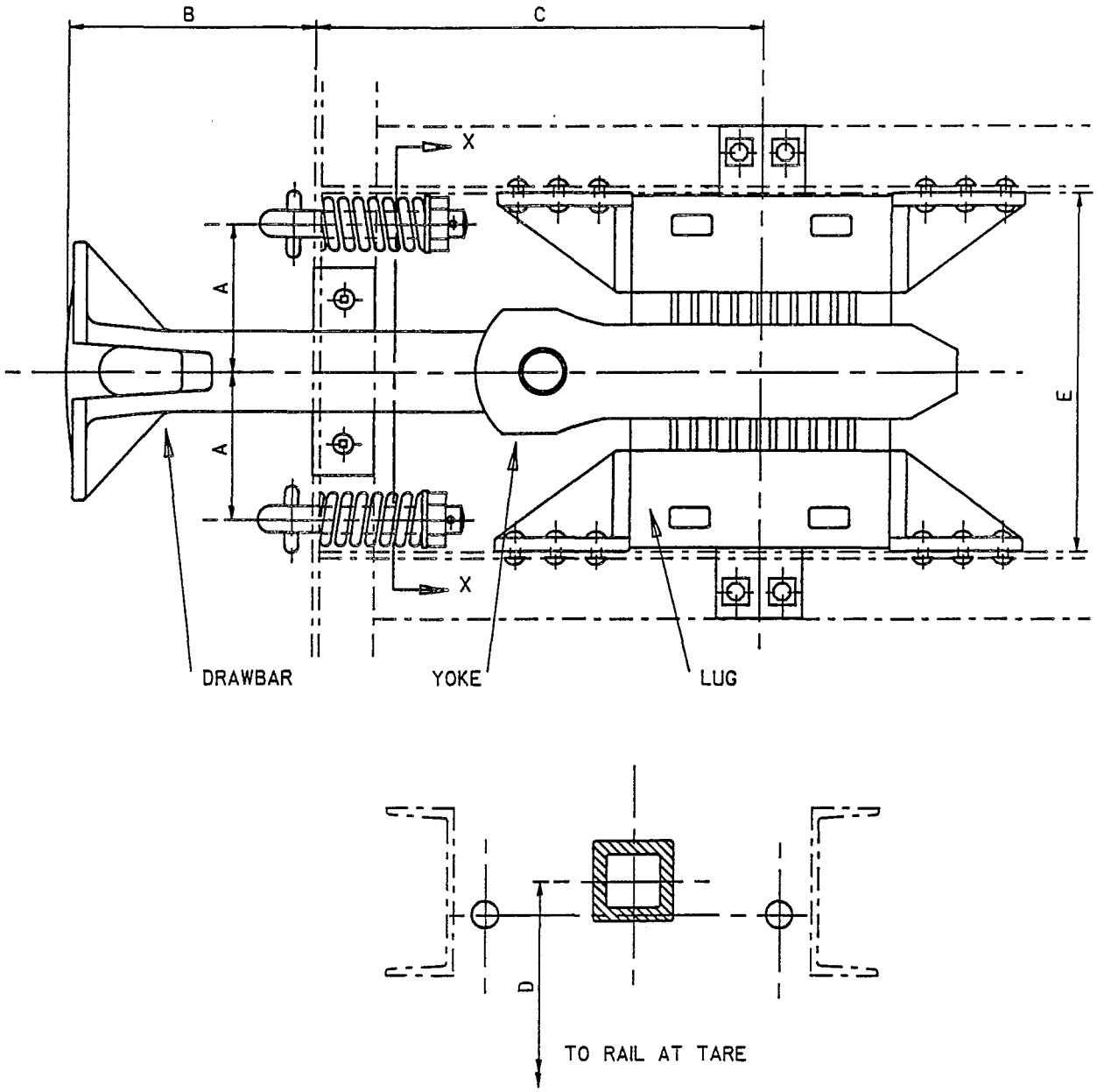
CHAPTER IV

WAGON DRAWGEAR DESIGN

1. DEFINITION, FUNCTION, AND PERFORMANCE

The drawgear is the assemblage that is used for connecting wagons and/or locomotives together and attenuating shocks associated with train operation and coupling of wagons. It must have some lateral and vertical movement to accommodate horizontal and vertical curves in the track, wear in wagon components and differing geometry of mating wagons. Compatibility between drawgear assemblies is not only required because of the need to couple wagons but also because of response to train movements and marshalling impacts. For wagons in a captive service the drawgear need only be compatible with other units in the same service and not with those in general service.

The service loadings and movements cause wear, fatigue, plastic deformations and permanent set in rubber components and fracture in drawgear components. Broken couplers and severe run-ins may cause derailments. From 1977 to 1980 New Zealand Railways had 12 derailments resulting from defective drawgear, 15% of all derailments caused by vehicle defects (Vink, 1981). The performance of the drawgear also affects train operation; the deficiencies and limits of the drawgear influence the operating practices of the locomotive drivers (train handling)



IDENTIFICATION	A	B	C	D	E	DRAWBAR	LUG	YOKE
28117	214	412	632.5	762	533	27907	31333	28084
28763	241	411	633	750	457	27907	27925	28084
27957	241	462	633	750	458	27907	27925	28084
28524	214	381	664	762	534	27907	28534	28084
29239	214	381	664	762	534	28992	28534	28084
29737	214	380	664.5	749	534	28992	31333	28084
28696	241	381	633	750	458	27907	27925	28084
28577	241	462	632.5	750	458	28576	27925	27925
31441	266.7	381	606	762	389	31440	27172/2	31374
28081	275	425	650	762	390	28096	31623	28084
26215	270	380	665	762	390	28992	31623	28084
32127	214	380	665	762	534	28992	31333	28084
32373	270	381	665	762	390	28992	31623	28084
29926	215	380	665	762	534	28992	31333	28084

Figure 8. Parameterised Wagon Drawgear Arrangement

and train make-up (particularly, the physical length of the train and the positions of loaded and empty wagons). From 1977 to 1980 13 derailments resulted from wagons being "pinched off" by accidental causes or error of judgement (Vink, 1981). Train dynamics is dealt with further in section 3 of this chapter .

Some of the qualities desired of drawgear are (West and others, 1978):

- transmission of low reactive forces by effective impact energy absorption;
- attenuation of force oscillations and high energy dissipation;
- durability;
- ease of "changeout" when maintenance is required;
- low capital cost;
- low drawgear slack and moderate travel.

2. DRAWGEAR SUBASSEMBLY DESIGN

The drawgear assembly may be further divided into two main subassemblies: the draftgear; and the coupler. Figure 8 presents some drawgear arrangements.

(1) Couplers and Drawbars

The coupler is the device for connecting wagons together. Two main types exist within NZR:

(a) Automatic Coupler. Automatic couplers are similar to the American couplers (which couple automatically by impact) and

are capable of being uncoupled from the side of the wagon. These have been applied to unit train wagons, and some wagons designed to carry dangerous goods, for example, the Ucg class, and LPG carrier, has a modified automatic coupler with vertical restraints to prevent vertical separation and possible puncture of the tank in a derailment. The limitations of the standard drawbar may result in the use of the automatic coupler in general service. The geometry and material of the coupler, in particular the coupling surfaces, are standard. The various standards and recommended practices (for example, Assoc. of American Railroads, 1982) also prescribe loads that the coupler must withstand.

American and European railroads have considered couplings that will automatically couple the brake pipe connection. There are also couplers which allow the wagon to be rotated about a longitudinal axis. These couplers are predominately used in service where bulk commodities are unloaded by this rotation of the wagon.

(b) Standard Drawbar. The standard drawbar with buffer head and drawbar hook is the coupling device that has been used on most NZR freight wagons. It has evolved over the years, modified to meet increased demands, for example, a larger hollow shank, and a wider buffer face, but there being essentially one design at any one time. The drawbar has a number of ancillary parts and assemblies most of which are used repeatedly, for example, bridle and pin assembly W32677, and drawhook X28082, and side chains which are selected from W/X28086/5.

There are some exceptions however, for example, although the

drawbar X28576 has the same head as X28992 it has a 50mm offset in the shank.

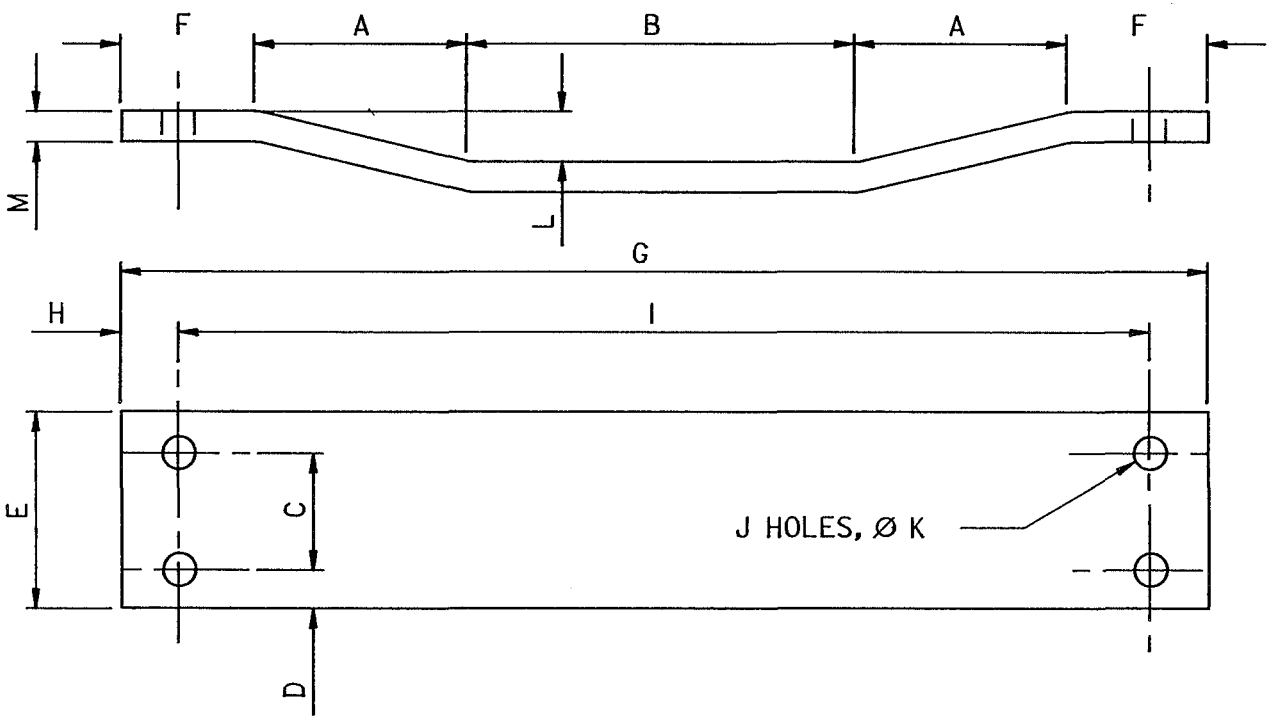
The wear limits (important for strength and coupler height considerations), heat treatment, and materials are laid down on the drawings, specifications and Mechanical Branch Workshop Code 45.

(2) Draftgear

The draftgear is the name of the unit which forms the connection between the coupler and the wagon solebars. The function of the unit is to absorb loads imposed by train action and coupling of wagons, and to protect freight, drawgear and wagon structure from excessive loads.

There are a number of types available that make use of friction or steel or rubber springs, or hydraulic damping, or some combination of these elements. They are typically rated by their maximum travel, reaction force, energy absorption and dissipation in both draft and buff. Long travel hydraulic cushioning units that require a specialised underframe and employed in US with reduction in freight damage claims have not been used in NZ.

The draftgear most frequently used in recent times in NZR is the Sumitomo type on Y36203 with spring blocks W31624, yoke X28084 and yoke pin Y/X28085/4, although higher gross weight bogie wagons have had other proprietary draftgear applied. The Sumitomo draftgear is located in a pocket formed by a yoke carrier plate and two cast steel draft lugs riveted or huck bolted to the solebars, whereas some other draftgears fit into a pocket formed by attachments welded to the solebars.



DRAWING	A	B	C	D	E	F	G	H	I	J	K	L	M
32433/1	60	430	76	26	128	81	712	38	636	4	26	32	20
28043/4-A	139	254	76	25	128	90	712	38	636	4	21.5	33	20
27773/1	70	203	64	38	203	127	597	38	521	6	20	9.5	16
29938/6	120	200	76	27	130	120	680	27	626	4	21	-12	20

Figure 9. Wagon Yoke Carrier Plates

If the connection was of a new design, it was checked for strength using methods such as presented in Assoc. of American Railroads (1982) for riveted or welded connections which prescribe edge distance and pitch limits, shear stress limits, and so on. Similar formulas have been used (NZR CMEO Design Office, C Wagon Design Notes) to calculate shear stress given shear area and vice versa. Other considerations as in normal bolted joint design were taken into account, for example, use of standard sizes, machining costs, symmetrical distribution of connectors, and so on.

In many cases the draft lugs can be selected from previous designs depending on solebar spacing and drawgear height relative to solebar height; for example, for draft lug W31333 the drawbar centreline cannot be less than 128mm from the top of the solebar or 70mm from the bottom of the solebar without special provision and must fit solebars spaced 534mm apart. Figure 9 presents some yoke carrier plates. Carrier plates, as well as other components, must retain structural integrity despite significant wear. (American practices allow 25% wear of carrier plates.)

3. TRAIN DYNAMICS AND WAGON IMPACTS

(1) Definition

Train dynamics is defined as the dynamic motion of a train and deals primarily with the study of the development of longitudinal coupler forces and track characteristics and their effects on vertical, lateral, and longitudinal vehicle movements and forces.

Excessive forces on the vehicle in a train or a marshalling yard impact may fracture or cause fatigue damage not only in drawgear components but in other vehicle parts such as axle boxes, brakegear, doors, endboards and underframe members. But more importantly in terms of satisfying the customers' transport needs, shocks transmitted through the coupler and coupling action which result in derailment can damage freight. Other problems studied include those of wagons being pulled off the inside of curves, lateral stability of the train, rail rollover, debogie-ing vehicles, and coupler overrides.

(2) Coupler Override

In America coupler overrides (when the coupler of one wagon vertically bypasses the coupler of the adjacent wagon) have resulted in the puncture of tank wagons carrying hazardous materials and in train crew fatalities. An investigation involving accidents, impact tests, and theoretical train-action models has led to the identification of several override mechanisms (Diboll and Peters, 1979). They claim coupler override occurs by either disengagement (when two mated couplers slide vertically with respect to each other and then uncouple), or vertical bypass (when two unmated couplers completely bypass each other). Coupler override is the result of yielded or fractured couplers, yielded solebars, wagon debogie-ing, derailing, wagon vertical dynamics, or a combination thereof. Diboll and Peters present a number of possible improvements to the vertical dynamics model in Raidt and others (1975) and adequately simulated a

squeeze override accident using the computer model in Yin (1976).

(3) Modelling Wagon Impacts

Earlier investigations have derived equations for relating, typically, the force between couplers and the change in potential energy to wagon masses, initial velocities, and draftgear characteristics when wagons impact solid stops or other wagons. For example, Freudenstein (1970) has presented an analysis based on lumped parameters including sliding friction that predicts peak forces experienced by the resilient freight for a constant force cushioning device. However more recent work has resulted in the development of computer-based simulation models with many more degrees of freedom and nonlinear behaviour of draft gears, suspension, couplers, freight, and so on (see, for example, Kasbekar and others, 1977). These models can be used for estimating the effect of wagon parameters and impact conditions on freight damage and to establish design criteria for drawgear and other wagon assemblies.

The work by Wilkinson (1978) on "pre- and postprocessors to handle the input-output problems" is particularly relevant to this thesis. As part of a project to model marshalling yard impacts, this work discussed programs to simplify the input of data selected from numerous combinations, ensure data checking and expedite examination of large quantities of output. The preprocessor is used to define the wagon string; the user selects the wagons from a "menu" of stored wagons (with input characteristics stored in a data file) and, if desired, changes

parameter values by designating the parameter with a cursor then entering the new value. The formulation and integration of the equations of motion is on a mainframe computer and is based on a previously published model. The postprocessor includes a module that presents the wagon movements in a motion picture format and a module for interactive plotting of selected variables with respect to time. The interactive pre- and postprocessors are all programmed for a graphics minicomputer. Wilkinson claimed that it is a simple matter to modify the programs to accommodate any alternative equation formulation and integration program.

(4) Draftgear Models

Various dynamic models for simulating nonlinear draftgear behaviour have been proposed (Hsu and Peters, 1978; Ward and Leonard, 1974; Kasbekar and others, 1977) based on experimental work, the resulting wagon impact models comparing favourably with test results. Surveys of marshalling yard speeds have also been performed (Diboll and Peters, 1979).

(5) Train Dynamics Models

(a) Longitudinal. Longitudinal train dynamics models compute as a function of time the forces in all couplers and the relative displacements of wagons arising from changes in locomotive control, track profile, brake operation, and friction forces. Earlier investigations modelled the train as a continuous bar or series of linear springs and masses to simulate longitudinal force transmission. Computer based models now

consider nonlinear draftgear behaviour, wagon weights, dimensions and positions, coupler slack, track grades and curves, braking system dynamics, and locomotive tractive effort. Several models have been developed employing various solution techniques. Genin and others (1974) used modal expansion to develop a simplified model in order to effect an economy of computer time, whereas Fitzgerald and others (1978) claimed a five fold improvement with their numerical integration method over that of a Runge Kutta method.

Longitudinal dynamics models have been used to investigate the effect of various factors, for example, slack, train length, draftgear characteristics and to assess features of the operating environment such as speed, track profile, and train handling procedures. Punwani (1980) has used the detailed train action model described by Martin and others (1976) to formulate a set of guidelines for improved in-train performance of draftgear units.

(b) Other Degrees of Freedom. Other models that include yaw and lateral degrees of freedom, such as in Garg and Tse (1978) and Yang and others (1972) can be used to investigate the effect of alignment control, wagon geometry, coupler length, and track geometry on lateral forces on the track and on coupler forces and to predict derailment conditions due to jackknifing. Formulas relating wagon geometry and coupler length, track geometry and drawbar pull to lateral forces are presented in Assoc. of American Railroads (1982) as is a method for checking horizontal and vertical curve negotiation. Assoc. of American Railroads (1982)

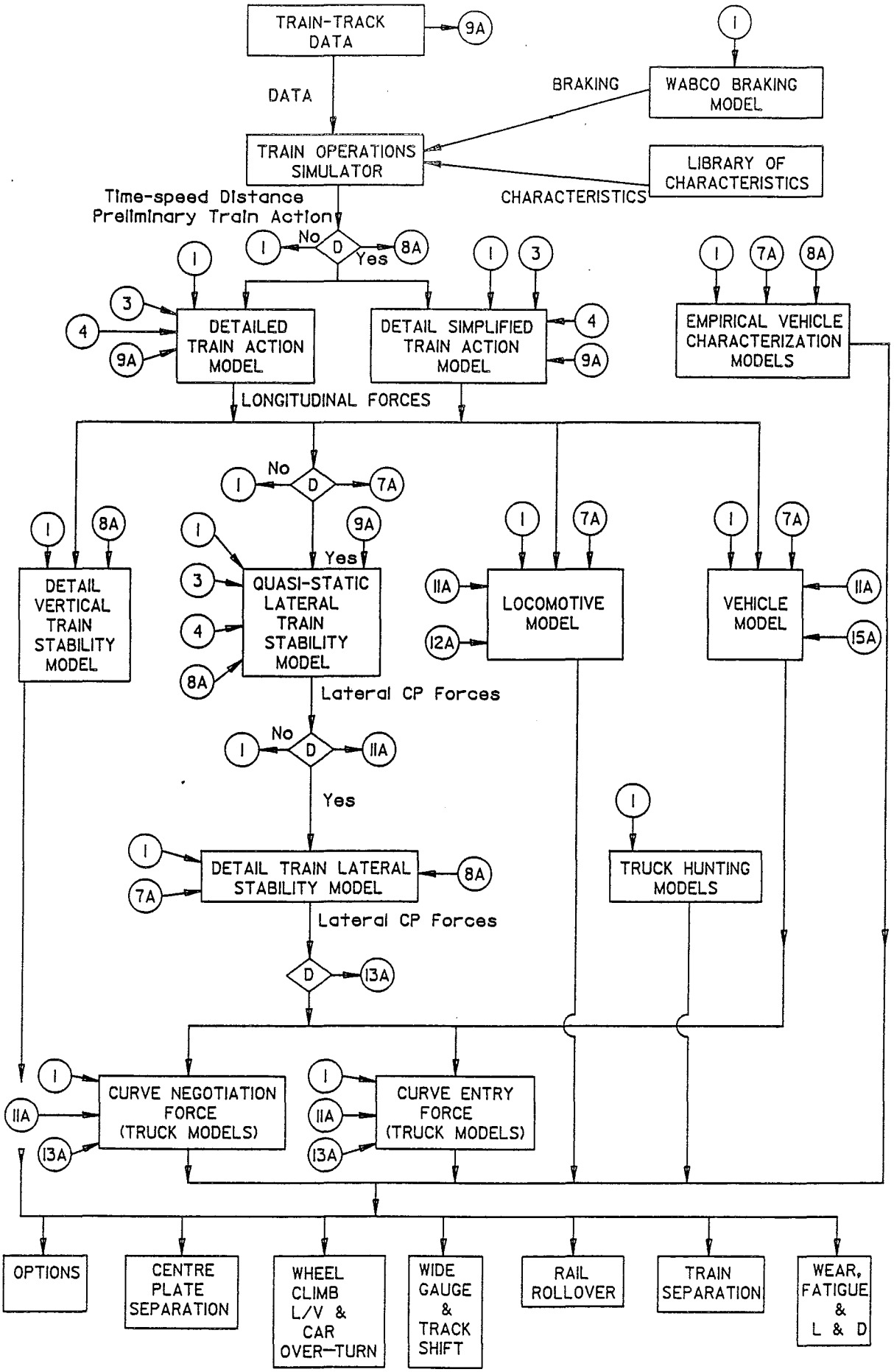


Figure 10. Information Processing of Models of Mathematical Modelling, TTD Program

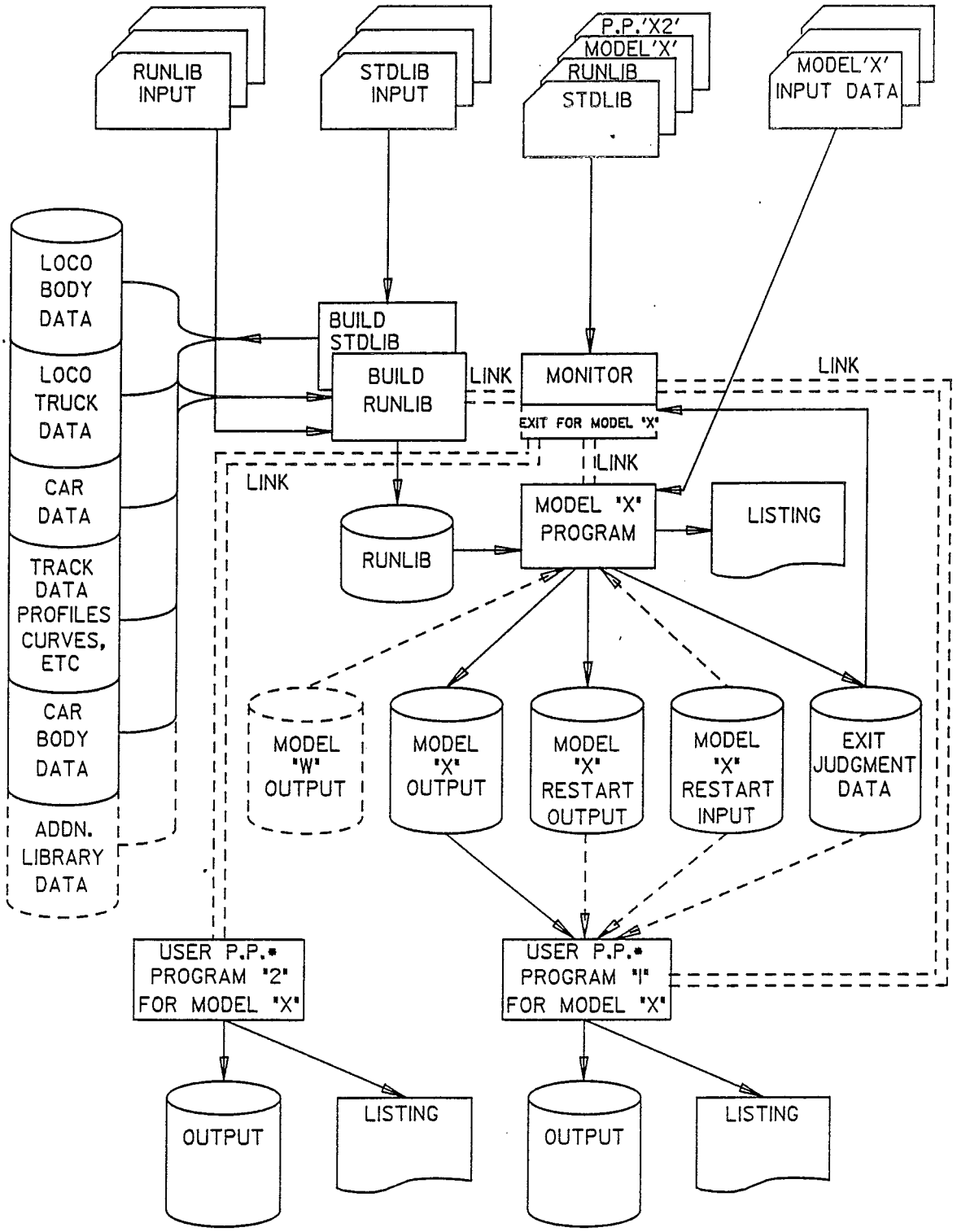


Figure 11. Pre- and Postprocessing and Monitor Concept, TTD Program

recommends the use of a horizontal curve negotiation program for calculating crossover negotiability.

The Assoc. of American Railroads Track Train Dynamics Program produced, in part, many computer based models. Rather than develop one general model that would be complex, costly to produce and difficult to use, they produced a number of smaller models which would address specific situations (Martin, 1975). The models were intended to be serially processed or used singly without reference to other models (refer fig. 10). For some problems, several models would give a solution but each with different characteristics (such as speed, level of detail, and accuracy). To automatically pass information between models and to alleviate the problem of large amounts of input data required they were developing a "library of characteristics" (containing such data as vehicle mass properties, springing and geometry), pre- and post processors (see sect. (3) for an example) and a monitor program (refer fig. 11). Recently these have been enhanced by addition of graphics and interactive features.

4. REFERENCES

ASSOC. OF AMERICAN RAILROADS. MECHANICAL DIVISION. (1982) Manual of Standards and Recommended Practices. Section C - Part II: Specifications for Design, Fabrication and Construction of Freight Cars. Vol. 1, Rev.ed.

DIBOLL, W.B. and PETERS, D.A. (1979) Coupler Override Mechanisms. ASME Paper No. 79-WA/RT-9.

FITZGERALD, B.W. and others. (1978) Application of Computer Simulation to Railroads. Proc. I.E. Aust. Conference on Heavy Haul Railways, Perth.

FREUDENSTEIN, F. (1970) Dynamic Analysis of Long-travel, High-efficiency Shock Absorbers in Freight Cars. Journal of Engineering for Industry, Trans. ASME, 92 : 581-589.

GARG, V.K. and TSE, Y.H. (1978) Mathematical Models for Track/train Dynamics. In Moyar, G.J. and others. Track/train Dynamics and Design : advanced techniques. New York, Pergamon Press Inc. p. 223-239.

GENIN, J. and others. (1974) Longitudinal Track-train Dynamics: a new approach. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 96 : 466-469.

HSU, T.K. and PETERS, D.A. (1978) A Simple Dynamic Model for Simulating Draftgear Behaviour in Railcar Impacts. Journal of Engineering for Industry, Trans. ASME, 100 : 492-496.

KASBEKAR, P.V. and others. (1977) Dynamic Simulation of Freight Car and Lading During Impact. Journal of Engineering for Industry, Trans. ASME, 99 : 859-866.

MARTIN, G.C. (1975) Description and Application of Track Train Dynamics Models. Rail Transportation Proc., ASME.

MARTIN, G.C. and others. (1976) User's Manual Detailed Longitudinal Train Action Model. Chicago, Illinois, AAR Technical Division. (AAR Research and Test Dept. Report

No. R-220).

NZR. CMEO Design Office. C Wagon Design Notes. (McDonald, I.) 1968.

PUNWANI, S.K. (1980) Draft Gear/Cushioning Unit Optimization for Train Action : final report, Vol. 1. Chicago, Illinois, AAR Technical Division. 115p. (AAR Research and Test Dept. Report No. R-363).

RAIDT, J.B. and others. (1975) Vertical Motions During Railcar Impact. ASME Paper No.75-WA/RT-10.

VINK, P. (1981) Derailment Statistics for Years 1979 and 1980. NZR Way and Works Branch Research Report 192.

WARD, E.D. and LEONARD, R.G. (1974) Automatic Parameter Identification Applied to a Railroad Car Dynamic Draft Gear Model. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 96 : 460-465.

WEST, L.E. and others. (1978) A Draft Gear Evaluation Project Applied to Heavy Unit Train Operations. Proc. I.E. Aust. Conference on Heavy Haul Railways, Perth.

WILKINSON, M.T. (1978) Real-time Modelling of Switch Yard Impacts. In Moyar, G.J. and others. Track/train Dynamics and Design : advanced techniques. New York, Pergamon Press Inc. p.261-269.

YANG, T.H. and others. (1972) Dynamic Analysis of Train Derailments. ASME Paper No.72-WA/RT-6.

YIN, S.K. (1976) Theoretical Manual and Users Guide : longitudinal - vertical train action model. Technical Report ORD - 76/278.

CHAPTER V

WAGON SUSPENSION DESIGN

1. DEFINITION, FUNCTION, AND PERFORMANCE

For the purposes of this thesis, the author has used the term vehicle suspension to denote the assembly of suspension elements connecting the wheelset bearings to the vehicle body.

(1) Function and Performance

Main functions of this assembly can be defined as (Wickens and Gilchrist, 1977):

(i) support the vehicle;

(ii) control the forces which guide the vehicle so that it follows the track with maximum steering ability, with proper entry into the curves and satisfactory negotiation of other track features;

(iii) filter out the effects of imperfections in track geometry, so that the vehicle experiences a smooth ride and the dynamic stresses applied to both the vehicle and the track are minimised;

(iv) stabilise the lateral motions of the vehicle so that sustained oscillation (usually referred to as hunting) does not occur at any speed within the operating range.

From a railway system point of view there is a trade off under fixed operating conditions between the cost of maintaining track condition with a certain level of track imperfections (and the initial investment in the track) and the capital investment and maintenance

costs of the vehicle suspension.

(2) Failure Modes

(a) Description of Failure Modes. Failure of the suspension to function ideally results in reduced safety and in damage to both the wagon and permanent way. Excessive lateral forces, particularly in curves, may increase rolling resistance, thereby increasing energy consumption, may increase noise levels and wear of flange and rail, and may also cause rail roll-over or gauge spreading. Severe hunting or bouncing may damage the lading and increase the rate of vehicle component and track wear and fatigue. Dynamic fluctuations in guidance and support forces can also adversely affect braking adhesion. For vehicles having a short wheel base and a flexible lateral suspension, longitudinal forces may result in relief of wheel load or vehicle yaw, that is, train buckling. On uniformly twisted track, the relief of vertical wheel load may lead to derailment. Vehicle roll (a problem more prevalent with North American high capacity, high centre of gravity wagons and staggered rail joints) results in wear and equipment failure and undesirable behaviour such as wheel-lift. It may also cause bottoming of suspension springs which may cause derailment and damage to car lading. High wheel forces at track joints, welds, switch and crossing work can cause rolling contact fatigue in the wheel tread as well as failures in the permanent way. Suspension component failure includes broken, cracked, or bent structural elements, worn or corroded components, parts knocked out of place, and interference with functional movement.

(b) NZR Derailment Statistics. Occurrences of derailments is one

measure of damage that is recorded by NZR. Derailments caused by defective springs, stays, hangers, axleboxes and hornguides, bogies, bolsters, and incorrect float limits, and by undamped oscillation of springs amounted to 29.7% of vehicle defect derailments between 1977 and 1980 inclusive (Vink, 1981). Vehicle/track dynamic interaction where track, vehicle and speed are individually within tolerance but have had a combined influence particularly when track faults are of a cyclic nature, resulted in 5.5% of all mainline derailments in the same period (Vink, 1981). (Vehicle defects in total accounted for 27.3% of mainline derailments.)

2. TYPES AND PARAMETERS

(1) Suspension Parameters

Once the decisions on the suspension concept have been made (for example, the number of axles per suspension assembly, and approximate location and layout of each unit), the values of the stiffness and damping parameters connecting the wheelsets to the wagon body and/or the bogie frame may be selected. The designer has influence over the initial conicity of the wheel treads but generally the effective conicity, creep/friction coefficients and such track characteristics as gauge, gradients, curvatures, cants, and track roughness lie outside the vehicle designers control and may have considerable variation. The range of payload masses and geometries, vehicle body mass, combined payload and body inertias and position of centre of mass, vehicle maximum speed, wheelbase, and wheelset inertias are mostly determined

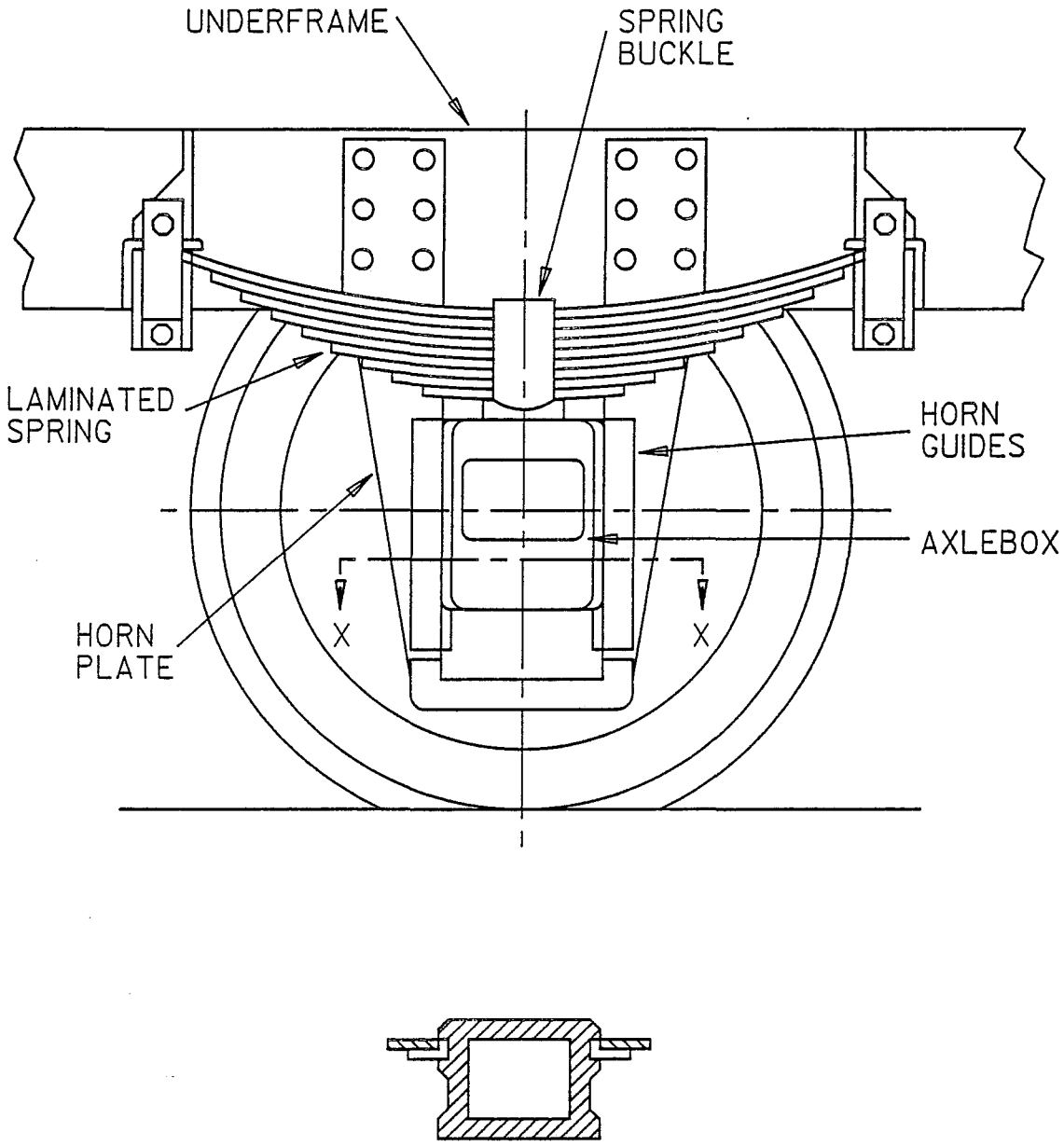


Figure 12. Shoe Suspension Arrangement

from the desired operating characteristics of the wagon over which the suspension designer only has some influence.

The selection of the parameters to achieve the suspension functions must make allowance for manufacturing tolerances, locomotive driver and yard worker error or lack of discipline, uneven load distribution, brake application and release and meet the reliability and maintenance specifications for the life of the vehicle.

(2) Suspension Types

Except for a well wagon (Ud, designed for heavy loads) with a three axled bogie, NZR freight wagons are equipped with either a pair of two axled "bogie" suspensions or a pair of single axled "four wheel" suspensions.

(a) Four Wheel Suspension. Until the mid-70's the standard four wheel suspension was the "shoe suspension". Conceived in the last century, it is a simple design with the body resting on the ends of laminated springs which react against the axlebox at their central buckle, the axlebox being constrained to move only in the vertical direction by guides or "horns" (refer fig.12). The lateral and longitudinal axlebox to horn clearances have specified maintenance tolerances.

Although with this type of suspension two-axled vehicles perform satisfactorily at relatively low speeds on good track, it has proved to be inadequate for modern operational requirements. This poor performance has been attributed to the linear vertical springing, unpredictable vertical friction and the lack of longitudinal and lateral springing and damping. Following a small increase in speed after metrication of train speeds in New Zealand, these four wheeled

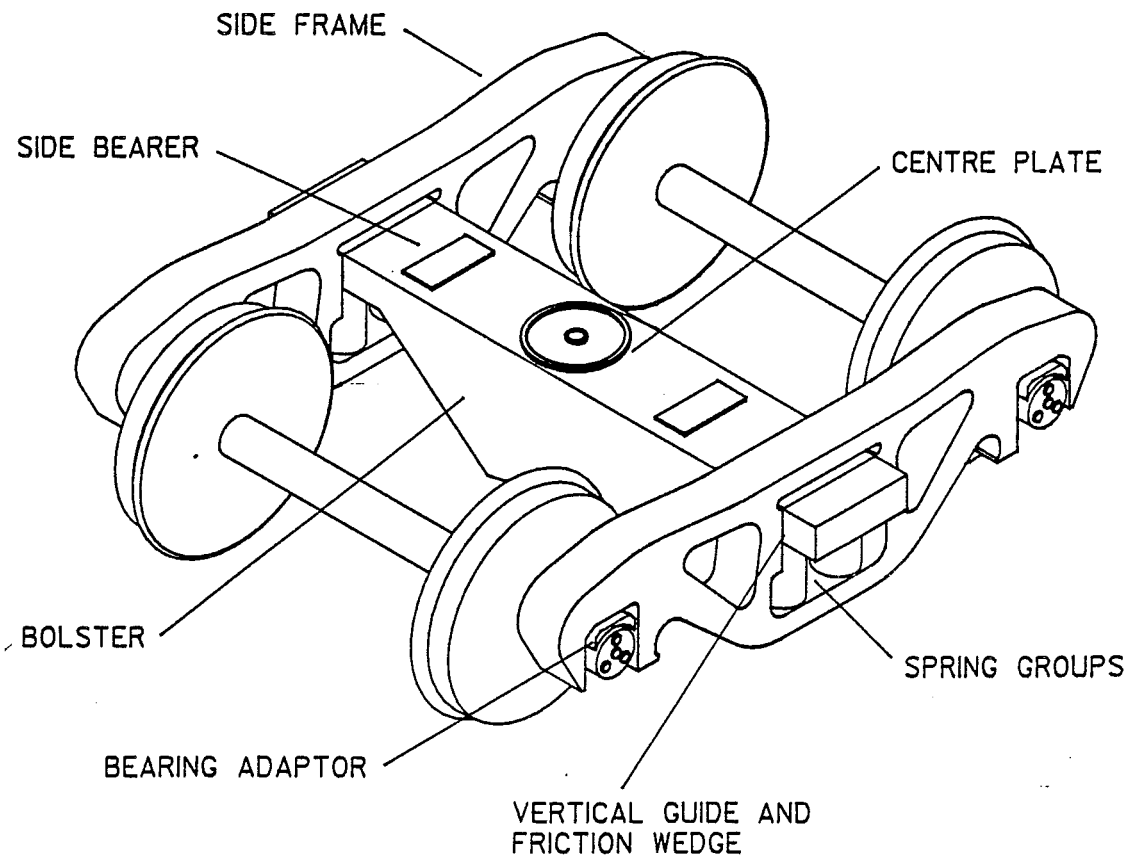


Figure 13. Bogie Suspension Arrangement

wagons had an unacceptable rate of derailment. This led to a reduction in operating speeds over rougher sections of track and the development of four wheel wagon suspension designs for express service, namely, the link suspension and the pedestal suspension.

Long links are provided in the link suspension to support the body on laminated springs. Larger lateral and longitudinal clearances between the axleboxes and horns are used to allow the links to act as damped centering devices.

The pedestal suspension is stiff longitudinally restraining the axles to be parallel. The pedestal acts as a swing link allowing lateral movement. Damping for both vertical and lateral motion is derived from the vertical load by using wedges and linered faces (for good wearing) hence damping is proportional to the payload.

Both suspensions have two rate vertical springing and, in NZR, have been tested and developed in a number of prototypes.

(b) Bogie Suspension. The three piece bogie has been for many years the standard bogie suspension in NZR, as it has been in many other countries. Its simple construction leads to ease of maintenance and a low first cost . At present the three piece bogie is available in NZR in two basic sizes for different rated maximum capacities (the type 14 and the type 16), but a number of variants exist. The bogie bolster is the beam between the two side frames which receives, through the centreplate (one of a pair of plates which fit into one another allowing the bogie to turn freely under the wagon), the weight of the wagon body and transfers it to the sideframes through springs on which it is carried (refer fig.13). The sideframes provide the vertical guides for the bolster and transfer the spring loads to the bearing adapters and

ultimately to the wheels. Both elements of the bogie frame are of cast steel. The springs are coil groups and damping is provided by friction wedge and linered faces. The ability of the side frames to pitch relative to each other gives good wheel load equalisation within the bogie.

However this design is not without its problems as has been documented in conference proceedings such as Heavy Haul Railways Conference (1978). Apart from high unsprung weight and poor ride quality, its unstable hunting and poor curve negotiation ability, accentuated by its freedom to lozenge or parallelogram, results in rapid wear of track and wagon components, and high tractive resistance. Recognition of these problems internationally has prompted a search for bogie suspensions offering a range of performance improvements. NZR is currently evaluating a number of proprietary designs as well as one designed by NZR staff with a rigid frame, primary springing and interwheelset links.

3. SUSPENSION DESIGN WITHIN NZR

As the author has not studied NZR suspension design in detail, the following paragraphs can only be regarded as a brief overview.

(1) Determination of Parameters

Suspension design within NZR (NZR CMEO Design Office : Lpa, Nx, Ksx, Yh, Nk, 14c Design Notes) has been performed using traditional methods such as those presented in Botham (1967); Batchelor and Jeffs (1966); Batchelor and Stride (1969); Koffman (1957). The natural frequencies of a simplified model (typically a mass with five uncoupled

degrees of freedom) have been placed so that either resonance (typically a sinusoidal forcing function with wavelength the same as the rail length) occurs at a speed well in excess of the maximum service speed or occurs at an operationally unimportant and sufficiently low speed. In either case damping was provided as a proportion of the critical damping coefficient, that proportion being based on experience.

To ensure proper matching of drawgear, the deflection under static loading was limited. For passive suspensions (that used under most if not all freight wagons) this competes with the requirement for twisted track negotiation.

To avoid low speed derailment while a vehicle is passing over twisted track, the static variation in wheel load over track with a specified twist has normally been constrained to limiting values. This requirement is more important for long wheelbase wagons. The high ratios of laden to unladen body masses of most modern wagons dictates the use of two rate springing in order to meet these requirements and usually nonlinear damping is also desirable in order to keep damping as a fraction of critical damping more or less constant.

Some of the other factors (that is, other than natural frequencies and twisted track negotiation) affecting the choice of wheelbase (and distance between bogie centres) have been bridge loadings and route negotiation, sensitivity to uneven loading, the ride at the wagon ends, interference with other assemblies particularly the drawgear and deep sectioned underframes, and past practice.

(2) Spring Design

The spring load-deflection characteristics and stresses have been

found using formulas relating deflection, stress, geometry, material properties and load. For example, those in Botham (1967) have been used for laminated springs and those in BS 1726 : Part 1 for helical springs. The Design Office have programmed their Hewlett-Packard 9825A calculator for laminated spring design; stiffnesses, stresses and deflections are printed out for specified spring geometry and loading. These relationships have also been used to determine, for example, the number of turns required for a helical spring of given stiffness, coil diameter and bar diameter. For laminated springs the stop distance (which prevents excessive deflection and acts as a safety measure in case of spring failure) and camber were determined for new spring designs. In the case of two stage springs, the geometry for correct change over has been found.

(3) Damping

The damping force required for the specified damping coefficient has been used to size the pins, in the case of the link suspension, and the wedge angle, in the case of the pedestal suspension, again using semi-empirical methods.

(4) Space Constraints

Space constraints limit the available geometries of the suspension assembly and components. For example, the bogie sideframes and wheels have to clear the vehicle body in horizontal and vertical curves in the loaded and empty condition, while the laminated spring in four wheel designs must not interfere with solebars or horn guides.

(5) Structural Components

The suspension structural components have been designed to endure suspension and braking loads in much the same way as the body structural components, but with assumptions and stressing models appropriate to the specialised components. More general engineering component design, such as that of welded and bolted joints, have also been required.

(6) Other Evaluations

Other evaluations that have been performed include the preparation of cost estimates and estimation of inertias of assemblies. The suspension system must allow access to all parts which require periodic inspection and maintenance.

4. INTERNATIONAL SUSPENSION AND VEHICLE DYNAMICS DESIGN

Internationally there has been substantial progress in the field of railway vehicle dynamics over the past twenty years or so as documented in the numerous publications (an earlier review is presented in Law and Cooperrider, 1974). As these publications are too extensive to go into detail, only some, judged to be the more important developments, will be described. Particularly noticeable are the contributions of British Railways, as evidenced by the presentation of the MacRobert Award of 1975 (Wickens and Gilchrist, 1977) to the British Railways Board in recognition of their achievements in this field, and the contributions of the Track/Train Dynamics Program, administered by the Association of American Railroads. The proven theories developed over this period permit not only detailed dynamic studies and new methods of parameter

specification in new and remedial design work, but have also stimulated innovations in suspension concepts like the high performance two-axled vehicles of the HSFV series (Wickens and others, 1969) and the cross-braced bogie (Mulcahy and Weeks, 1982; Pollard, 1979).

(1) The Response to Track Irregularities

The response of vehicles to irregularities of the track has had an important influence on the selection of suspension stiffnesses and damping in these studies (Wickens and others,1969). Suspension forces and displacements for stressing and suspension travel, wagon body accelerations, wheel/rail forces and relative motion are some of the parameters of interest.

To characterise track roughness the vertical profile of the track centre-line, the lateral, and the cross-level profiles have been measured by a variety of methods. Track gauge and rail head profile are required to assess conicity and contact angle parameters and equilibrium rolling line, but have not been able to be measured continuously by past methods.

The equations of motion have been adequately covered elsewhere in the relevant publications and require no special considerations except for the creep, conicity and contact stiffness terms and when track dynamic behaviour is required. The vehicle can be treated as a spring mass system with parts connected by triaxial spring damper elements.

Continuously irregular track has often been characterised by its power spectral density which specifies the intensity of the irregularities as a function of spatial frequency. Random process theory allows the calculation of the vehicle response in corresponding

terms. Realistic track spectra have prominent periodic and almost periodic inputs, especially those of the different harmonics of the rail length, in addition to wide band random inputs (Newland and Cassidy, 1975; Elmadany and Ramachandran, 1981).

The equations have also been solved by integration often when considering response to an idealised discrete irregularity, for example, a "dipped rail joint" (Jenkins and others, 1974).

A number of computer programs with varying modelling detail and solution methods have been developed to study roll and bounce behaviour of North American freight wagons (Garg and Tse, 1978).

(2) Lateral Stability

Researchers and designers have used a series of mathematical models and associated solution techniques to study the lateral stability of rail vehicles. These studies are concerned with the question of whether the free motion of the vehicle at some forward speed, following a small initial disturbance, will decay or grow. Generally the simultaneous differential equations are derived by assuming a perfectly aligned straight and rigid track, and negligible braking and aerodynamic forces (Wickens and Gilchrist, 1977; Law and Cooperrider, 1974).

At a sufficiently high speed above the boundary of critical speed the lateral oscillations grow, eventually to be limited by nonlinear effects. A sufficient stability margin can be achieved in simple terms in new designs by placing the critical speed above a tolerance on the maximum operating speed. If it is not achieved there may be a number of modes of instability: body instabilities when the frequency of the wheelset kinematic movements coincide with the natural frequencies of

the body on the suspension and associated with larger body motion; bogie hunting in which the body movements are relatively small and motions of the bogie frame sustain wheelset motion; and wheelset instability, usually associated with the highest critical speeds in which motion is largely confined to the wheelsets (Bell and others, 1981). Body instabilities may be eliminated by the appropriate selection of stiffnesses and damping for the range of effective conicities and creep coefficients expected in service (Wickens and others, 1969), while the critical speed of bogie instabilities may be raised by modest primary stiffness and/or friction at the pivot (Mulcahy and Weeks, 1982).

The stability of a vehicle may be predicted by finding the eigenvalues (frequencies and damping factors) and eigenvectors (mode shape - the relative magnitudes of the component's motion) of the characteristic determinant of the equations of motion. Plots can then be made of the root locus as a system parameter (for example, speed) varies, of stability boundary (contour of zero damping) or of eigenvalues against one or more system parameters and so on. Alternatively the differential equations can be solved by numerical or analogue integration for a specified set of initial conditions. The time histories of forces and displacements can be inspected for indications of instability or limit cycle oscillations.

For symmetric vehicles and small displacements the "lateral" or asymmetric (and non-conservative) group of equations separate from the "vertical" or symmetric group (those relating the motions in the vehicle longitudinal/vertical centre-plane, for example, bounce and pitch) (see for example, Wickens and others, 1969; Hobbs and Pearce, 1974). The solutions from the linear lateral simultaneous equations of motion have

been consistent with model and full scale experiments particularly with regard to the effects of the various parameters on stability. However selection of parameter values may be crucial, requiring a considerable amount of judgement (Tse and others, 1979; Wickens and Gilchrist, 1977; Cooperrider and Law, 1978; and Cooperrider, 1983). Predicted behaviour for special cases using this simple linear theory has aided understanding of particular details of the mechanism of instability, but recourse to computer simulation is necessary for detailed studies (Wickens, 1965).

The effect of various complicating factors such as body and axle flexibility (Hadden and Law, 1977), asymmetric loading, axles with different wheel profiles (Tuten and others, 1979) and adjacent wagons have been investigated (Blader and Kurtz, 1974).

Nonlinear features such as nonlinear suspension elements, nonlinear wheel/rail contact geometry and creep forces have also been modelled (for example, Tse and others, 1979; D'Souza and Caravavatna, 1980; Horak and Wormley, 1982; Burton, 1981). A number of studies have employed various forms of equivalent linearisation of nonlinearities including describing functions (for example, see Hull and Cooperrider, 1977), and statistical linearisation (for example, Hendrick and Arslan, 1979).

Generally it is possible to obtain good predictions of critical speed and lateral and vertical response, but the validity of calculations to predict critical speed and lateral response of stiff and very nonlinear (in particular, high levels of friction) vehicles remains unproven and it has been suggested that the use of linear transfer functions to predict the lateral response of three piece freight bogies may be inappropriate (Gostling and Cooperrider, 1983).

(3) Curve Negotiation

The steady state geometry and force distribution of a vehicle in a perfect circular curve and the dynamic response of a vehicle due to curve and switch entry have been studied. The nonlinear steady state curving theory published by Elkins and Gostling (1977) which included the effects of the real cross-sectional geometry of wheels and rails and saturation of the creep force/creepage relationship has recently been improved to allow traction and braking torques to be applied to the wheelset, calculation of the power required to propel the wheelset and the creep force/creepage data has been extended to allow prediction of behaviour up to and including derailment (Elkins and Eickhoff, 1982). Experiments have demonstrated that the nonlinear curving theory can accurately predict wheel/rail forces and wheelset attitudes for a wide range of curves and soft or stiff vehicle suspensions (Elkins and Eickhoff, 1982) This work has also demonstrated that the classical ideas of curving presented by Porter (1935) are applicable in sharp curves where creepage is in the full slip regime and that the linear theories published by Boocock (1969) and Newland (1969) describe the situation for large radius curves. The nonlinear theory is being used to predict forces for stressing purposes and to calculate wear indices in order to rank vehicle curving behaviour (Elkins and Eickhoff, 1982). It promises to be a valuable tool in the evaluation of future designs of vehicles and suspensions. There are also a number of other mathematical models, for example, see Garg and Tse (1978).

(4) New Suspension Concepts

Wickens has investigated the theoretical possibilities of unsymmetric two axle vehicles (Wickens, 1982) and general forms of interconnections between wheelsets, bogie frames and bodies (Wickens, 1978) in the search for an optimum compromise between the requirements of stability and curving.

(5) Validation of Theories

Cooperrider and Law (1978) presented a survey of validation efforts which included progress on track and vehicle characterisation, and a survey of solution techniques applied to vehicle curving, stability, random response, and roll behaviour. The survey of lateral dynamics validation and characterisation efforts was updated by Gostling and Cooperrider (1983). In Cooperrider and others (1978) an hierarchy of curving, stability and response models with varying degrees of detail and using different solution techniques was reported. The computed results were being compared with test results to establish the conditions and range of applicability of each model and solution technique so that eventually the vehicle designer can select the computer program that is appropriate for the purpose at hand.

(6) Optimisation of Suspension Design

Optimisation algorithms have been employed in suspension studies. Samaha and Sankar (1980) used nonlinear optimisation techniques to determine suspension parameters that minimise the maximum rocking response. Cox and others (1978) obtained the maximum operating speed (defined as a fixed percentage of the hunting speed) with various

suspension parameters as independent variables. Their linear dynamic model was subject to random alignment irregularities and constraints on maximum car body accelerations, maximum suspension travels, maximum creep forces that precluded gross sliding over the entire contact area, and maximum secondary suspension travel in curves.

(7) Wheel Rail Contact

Creep and conicity give the wheelset its guidance and its propensity to instability.

(a) Geometry. The geometrical aspects of wheel/rail rolling contact influence the behaviour of the contact point on each wheel as the wheelset is displaced relative to the track. Analytical solutions to the effect of lateral displacement on various parameters for purely coned wheelsets, and for rail and wheel cross-sectional profiles represented by circular arcs are available but in practice profiles are of continuously varying curvature. Some tyre profiles have been designed to resemble more closely a worn shape for reduction in maintenance and for stability and curving requirements. Various wheel and rail profile measuring machines have been developed; the results are often stored in digitised form on a computer system. Computer programs have been developed to determine the wheel/rail geometric relationships from the mutual contacting geometry when the profiles of a particular wheel and rail pair are combined (Cooperrider and Law, 1978; Gostling and Cooperrider, 1983). The rolling radius difference (the slope of the rolling radius difference versus lateral displacement graph is defined as equivalent conicity), contact angles, size and shape of contact patches computed for different wheel set lateral positions have been

stored in data tables (Elkins and Gostling, 1977). Several effects of the nonlinear wheel/rail contact geometry on the wheelset motion, for example, the contribution of wheelset yaw (Tse and others, 1979), have usually be neglected (Burton, 1981).

(b) Creep. Many workers have contributed to the theories relating tangential forces to wheelset motion but Kalker's creep force/creepage theories have been confirmed by laboratory experiment and are regarded as the most complete theories available (Kalker, 1979). These theories predict the lateral and longitudinal forces and spin moment due to the relative lateral, longitudinal and angular velocities between wheel and rail (that is, the creepages). For small creepage the force/creepage relations are linear and there is an area of relative slipping within the contact region, while at very large creepage the whole contact region degenerates to the case of pure sliding. In practice Kalker's coefficients and forces are often factored down, in line with the reduction due to surface contamination found in track tests (refer chapter VI, sect. 7 for discussion on adhesion levels), although this assumption has been questioned (Gostling and Cooperrider, 1983). The creep coefficients are available in tabular form and from computer programs developed for calculating creep forces according to both the simplified theory and exact theory (Kalker, 1979; Kalker, 1982).

(8) Wheelclimb Derailment

These advances in information on rolling contact phenomena has enabled more detailed studies into wheelclimb derailment conditions, that is, when one wheel of a laterally loaded wheelset climbs up and

over the rail while rolling forward and being pressed against the rail (Elkins and Eickhoff, 1982; Sweet and Karmel, 1981).

5. REFERENCES

BATCHELOR, G.H. and JEFFS, D.C. (1966) The Riding Properties of 4-wheeled Wagons. British Railways Board. Report No. T-27.

BATCHELOR, G.H. and STRIDE, R.C.T. (1969) Hydraulic Dampers and Damping. Journal of the Institution of Locomotive Engineers, 58 : 563-629.

BELL, C.E. and others. (1981) Stability and Curving Mechanics of Rail Vehicles. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 103 : 181-190.

BLADER, F.B. and KURTZ, E.F. (1974) Dynamic Stability of Cars in Long Freight Trains. Journal of Engineering for Industry, Trans. ASME, 96 : 1159-1167.

BOOCOCK, D. (1969) Steady-state Motion of Railway Vehicles on Curved Track. Journal Mechanical Engineering Science, 11 (6) : 556-566.

BOTHAM, J.M. (1967) Practical Aspects of Primary Suspension Design. Journal of the Institution of Locomotive Engineers, 57 : 495-535.

BRITISH STANDARDS INSTITUTION. (1964) Helical Compression Springs. Pt.1. 36p. (BS 1726: 1964).

BURTON, T.D. (1981) Influence of Wheel/rail Contact Geometry on Large Amplitude Wheelset Equations of Motion. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 103 :

211-218.

COOPERRIDER, N.K. and LAW, E.H. (1978) A Survey of Rail Vehicle Testing for Validation of Theoretical Dynamic Analyses. Journal of Dynamic Systems, Measurement, and Control, Trans.ASME, 100 : 238-251.

COOPERRIDER, N.K. and others. (1978) Theoretical and Experimental Research on Freight Car Lateral Dynamics. Proc. I.E. Aust. Conference on Heavy Haul Railways, Perth.

COX, J.J. and others. (1978) Optimization of Rail Vehicle Operating Speed with Practical Constraints. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 100 : 260-269.

D'SOUZA, A.F. and CARAVAVATNA, P. (1980) Analysis of Nonlinear Hunting Vibrations of Rail Vehicle Trucks. Journal of Mechanical Design, Trans. ASME, 102 : 77-85.

ELKINS, J.A. and EICKHOFF, B.M. (1982) Advances in Nonlinear Wheel/rail Force Prediction Methods and Their Validation. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 104 : 133-142.

ELKINS J.A. and GOSTLING, R.J. (1977) A General Quasi-static Curving Theory for Railway Vehicles. Proceedings 5th VSD 2nd IUTAM Symposium, Vienna.

ELMADANY, M.M. and RAMACHANDRAN, P.V. (1981) Lateral Stability of Flat Rail Cars : an over-the-road investigation. Int. Journal of Vehicle Design, 2 (2) : 162-173.

GARG, V.K. and TSE, Y.H. (1978) Mathematical Models for Track/train Dynamics. In Moyar, G.J. and others. Track/train Dynamics and Design: advanced techniques. New York, Pergamon Press Inc.

p.223-239.

GOSTLING, R.J. and COOPERRIDER, N.K. (1983) Validation of Railway Vehicle Lateral Dynamics Models. Vehicle System Dynamics, 12 : 179-202.

HADDEN, J.A. and LAW, E.H. (1977) Effects of Truck Design on Hunting Stability of Railway Vehicles. Journal of Engineering for Industry, Trans. ASME, 99 : 162-171.

HENDRICK, J.K. and ARSLAN, A.V. (1979) Nonlinear Analysis of Rail Vehicle Forced Lateral Response and Stability. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 101 (3) : 230-237.

HOBBS, A.E.W. and PEARCE, T.G. (1974) The Lateral Dynamics of the Linear Induction Motor Test Vehicle. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 96 : 147-157.

HORAK, D. and WORMLEY, D.N. (1982) Nonlinear Stability and Tracking of Rail Passenger Trucks. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 104 : 256-263.

HULL, R. and COOPERRIDER, N.K. (1977) Influence of Nonlinear Wheel/rail Contact Geometry on Stability of Rail Vehicles. Journal of Engineering for Industry, Trans. ASME, 99 : 172-185.

JENKINS, H.H., and others. (1974) The Effect of Track and Vehicle Parameters on Wheel/rail Vertical Dynamic Forces. Railway Engineering Journal, 3 (1) : 2-26.

KALKER J.J. (1979) Survey of Wheel-rail Rolling Contact Theory. Vehicle System Dynamics, 5 : 317-358.

KALKER, J.J. (1982) A Fast Algorithm for the Simplified Theory of Rolling Contact. Vehicle System Dynamics, 11 (1) : 1-13.

KOFFMAN, J.L. (1957) Vibrational Aspects of Bogie Design. Journal of the Institution of Locomotive Engineers, 47 : 549-634.

LAW, E.H. and COOPERRIDER, N.K. (1974) A Survey of Railway Vehicle Dynamics Research. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 96 : 132-146.

MULCAHY, H.W. and WEEKS, R.J. (1982) Development of a Cross-braced Truck. Journal of Engineering for Industry, Trans. ASME, 104 : 182-189.

NEWLAND, D.E. (1969) Steering a Flexible Railway Truck on Curved Track. Journal of Engineering for Industry, Trans. ASME, 91 (3) : 908-918.

NEWLAND, D.E. and CASSIDY, R.J. (1975) Suspension and Structure : some fundamental design considerations for railway vehicles. Railway Engineering Journal, 4(2) : 4-34.

NZR. CMEO Design Office. Lpa Prototype and Four Wheel Link Suspension Design Notes. (Silvester, J.M. and May, D.J.) 1972 and 1974.

NZR. CMEO Design Office. Nk Wagon Suspension Design Notes. [1980].

NZR. CMEO Design Office. Nx, Ksx Pedestal Suspension Design and Development Notes. (May, D.J.) [1976].

NZR. CMEO Design Office. Yh Wagon Design Notes. (Atken, M) 1973.

NZR. CMEO Design Office. 14C Bogie Design Notes. (Cannell, A.E.) 1978.

POLLARD, M.G. (1979) The Development of Cross-braced Freight Bogies. Rail International, 10(9) : 736-758.

PORTER, S.R.M. (1935) The Mechanics of a Locomotive on Curved

Track. London, The Railway Gazette.

SAMAHA, M. and SANKAR, T.S. (1980) Dynamic Rocking Response and Optimization of the Nonlinear Suspension of a Railroad Freight Car. Journal of Mechanical Design, Trans. ASME, 102 : 86-93.

SWEET, L.M. and KARMEL, A. (1981) Evaluation of Time-duration Dependent Wheel Load Criteria for Wheelclimb Derailment. Journal of Dynamic Systems, Measurement, and Control, Trans. ASME, 103 : 219-227.

TSE, Y. and others. (1979) Validation of Freight Car Hunting (Nonlinear/Linear) Model. Chicago, Illinois, AAR Technical Division. 156p.(AAR Research and Test Dept. Report No. R-324).

TUTEN, J.M. and others. (1979) Lateral Stability of Freight Cars with Axles Having Different Wheel Profiles and Asymmetric Loading. Journal of Engineering for Industry, Trans. ASME, 101 : 1-16.

VINK, P. (1981) Derailment Statistics for Years 1979 and 1980. NZR Way and Works Branch Research Report 192.

WICKENS, A.H. (1965) The Dynamics of Railway Vehicles on Straight Track : fundamental considerations of lateral stability. Proc. Institution of Mechanical Engineers, 180 (Part 3F) : 29-44.

WICKENS, A.H. (1975) Steering and Dynamic Stability of Railway Vehicles. Vehicle System Dynamics, 5 : 15-46.

WICKENS, A.H. (1977) Static and Dynamic Stability of a Class of Three-axle Railway Vehicles Possessing Perfect Steering. Vehicle System Dynamics, 6 : 1-19.

WICKENS, A.H. (1978) Stability Criteria for Articulated Railway Vehicles Possessing Perfect Steering. Vehicle System Dynamics, 7 : 165-182.

WICKENS, A.H. (1982) Static and Dynamic Stability of Unsymmetric Two-axled Railway Vehicles Possessing Perfect Steering. Vehicle System Dynamics, 11 : 89-106.

WICKENS, A.H. and GILCHRIST, A.O. (1977) Railway Vehicle Dynamics - the emergence of a practical theory. Council of Engineering Institutions MacRobert Award Lecture, 1977.

WICKENS, A.H. and others. (1969) Suspension Design for High-performance Two-axle Freight Vehicles. Proc. Institution of Mechanical Engineers , 184 (Part 3D) : 22-36.

CHAPTER VI

WAGON BRAKE DESIGN

1. DEFINITION, FUNCTION, AND PERFORMANCE

Most modern wagons have a parking brake or handbrake and an air brake such that when the airbrakes of the locomotives and wagons in a train are connected the train brake is formed. The handbrake is an arrangement of levers, rods, gears, fulcrums, and so on, actuated manually by a wheel or lever and used to force the brake blocks against the braking surfaces to hold the wagons in a state of rest when applied. The air brake is the combination of parts operated by compressed air and controlled pneumatically (although brake systems exist where it is controlled hydraulically or electrically but not in NZ) that control the means by which the speed of a wagon is retarded or arrested.

(1) Performance Issues

The demand to restrict vehicle mass (for fuel saving) and particularly the unsprung components (to minimise track damage) has an important influence on brake system design. There is also greater commercial pressure to minimise brake system life cycle costs. In British Rail freight vehicle braking can account for 25% of the total capital and maintenance costs of the vehicle over its life (Waldron, 1979). Blaine and others (1981) present some recent costings for

American railroads, which tell a similar story. Safety is also an important issue, dictating reliable operation under the worst conditions.

(2) NZR Derailment Statistics

Broken or detached brake rigging is a derailment hazard. From 1977 to 1980 New Zealand Railways had 6% of all derailments caused by vehicle defects attributed to brakes and braking gear defects (Vink, 1981). The sudden loss of air pressure as a result of a ruptured air hose has also caused derailments. Brakes that fail to release can result in inability to maintain schedules as well as drawgear, brakegear and wheel failures.

(3) Brake Application

A braking application consists of the following stages: human reaction to the situation, initiation time from movement of the locomotive controls to movement of the first brake shoes, propagation as each vehicle's airbrake senses a reduction in train pipe pressure, a rise in application to the level desired, constant application, initiation and propagation of release as the control valves sense a rise in train pipe pressure, recharge of reservoirs, and finally full release. It is these propagation and application rates which can result in differences in vehicle velocities and transmission of shocks along the train. There have been computer programs developed to model the performance of the brake system in a complete train. The program developed by Hart and Grejda (1972) is used in the AAR train dynamics programs. Computer models predicting train behaviour underbraking are discussed in chapter IV . Models of the pneumatic part of the train

brake have been used to investigate the effect of leakage distribution on brake pipe pressure gradient and brake pipe air flow (Shute and others, 1979).

The train brake may be applied in an emergency when it must stop the train in the shortest time and distance with minimal wheel skidding and damage to freight and vehicles. Or it may be applied in normal service in situations such as stopping for stations and signals, for speed restrictions, retardation of speed on gradients, and control of yard movements.

2. THE BRAKE ARRANGEMENT : SUBASSEMBLY TYPES AND SELECTION

If approximate vehicle weights, structural layout and suspension type are known then the selection of the functional units that make up the brake systems can proceed. The pneumatic gear consists of pipes, hoses, control valves, cocks, reservoirs, cylinders and so on, assembled so as to produce the braking forces and modulate them in response to actuating signals. The brake rigging is the levers, rods, brake beams and so on by which the piston rod at the brake cylinder is connected to the brake blocks in such a manner as to transmit and multiply the forces developed by the cylinder (see fig. 15 and 17). The brake blocks transform kinetic energy into heat at the friction surfaces.

(1) Types of Brake Systems

Most of the NZR fleet is equipped with a single brake cylinder, which uses a lever system that multiplies force via fulcrum points, although there are some two cylinder systems. There are also available

tread actuation systems (in which the actuation device acts directly on the brake block head) and multiple cylinder direct acting types (with one or more cylinders mounted in a bogie). Instead of the brake block acting on the tread, it may act on a disc on the axle or wheel.

In NZR disk brake systems have been used mainly in trial service and consist mostly of proprietary equipment. Worldwide there has been substantial experience gained in various construction types, materials and services (see, for example, Tickle, 1979; and Anon, 1979).

(2) Economic Evaluation

The economic evaluation of brake system components/assemblies has sometimes only involved an investigation into the initial costs for a range of components/assemblies or, if historical costings were available, evaluation may have involved methods such as net present value. Typically the information required includes : vehicle usage; component life or meantime to failure; cost of replacement parts; cost of transport of replacement and scrapped parts; scrap part values; maintenance (labour and overhead) costs; wear debris related costs; lost revenue due to being out of service for repair; marshalling yard costs per failure; other than wagon operating costs (such as the cost of compressed air supplied by the locomotive); inspection and testing costs; and so on. But other factors such as ease of access, ease of removal of subassemblies, workshop maintenance capability, rail and wheel skidding, the effect of changes in operational practices on wear, and so forth, required, for the most part, human judgement.

(3) Pneumatic Equipment

The brake pipe is that section of the air brake piping which acts as a supply pipe for the reservoirs and, in full pneumatic brakes, is also the means of controlling application and release of the brakes. At the ends flexible hoses provide the connection between the wagons. At the brake pipe end, the airbrake hose is attached to the angle cock which is used to open or close the brake pipe, and the other end is fitted with a coupling which engages with a coupling on the adjoining wagon. The control valve (or triple valve) is a three way valve which charges the reservoirs, admits air to the brake cylinder, and releases air from the brake cylinder, in response to a reduction or increase of brake pipe pressure. The auxiliary reservoir is a storage structure for compressed air, charged from the brake pipe, which provides air pressure for use in service and emergency brake applications. The maximum application is when the auxiliary reservoir and brake cylinder pressures are equalised, otherwise the auxiliary reservoir pressure is reduced by venting to the brake cylinder to balance that of the train pipe, hence pressure in the cylinder is proportional to the reduction in pressure in the train pipe. The application and release timings are controlled by chokes and the internal volume of the auxiliary reservoir, piping, cylinder, and so on, thereby providing a margin of stability for pneumatic gear operation.

In an effort to improve release times and exhaustability, bigger brake pipes, two pipes as the "brake pipe", graduated release control valves, and electro-pneumatic systems, have been used by overseas railway organisations. But within NZR the need to be compatible with existing stock (particularly general service stock but less so with unit

train service stock) means significant changes have similar implications to those introduced when considering changing coupler types. Thus most items of pneumatic gear are essentially standard, supplied by manufacturers to meet agreed performance standards (for example, see Assoc. of American Railroads, 1978a). For some items there may be a range of products such as the W triple valve range where, for example, the WF2 triple valve has, amongst other features, an accelerated release feature, which functions to admit air from the accelerated release reservoir into the brake pipe effecting faster release of control valves in the train. These components and the other standard items such as dust collectors and release valves are superseded from time to time with improved product ranges.

(4) Brake Rigging Equipment

A common component of the rigging in recent wagons has been the double acting automatic slack adjuster. Its duty is to maintain a prescribed brake cylinder piston travel and is selected from a range of proprietary products partly on the basis of its capacity to absorb the travel due to the wheel and brake block wear. Other components such as pins, levers (which transmit and multiply forces from pinned joint to pinned joint or pinned joint to shaft), pullrods and pushrods (transmission of tensile and compressive forces respectively), brackets and slides, have varying geometry to suit the particular application.

(5) Load Compensating Equipment

Load compensating equipment varies the braking force for a given brake pipe pressure reduction according to the weight of the vehicle's

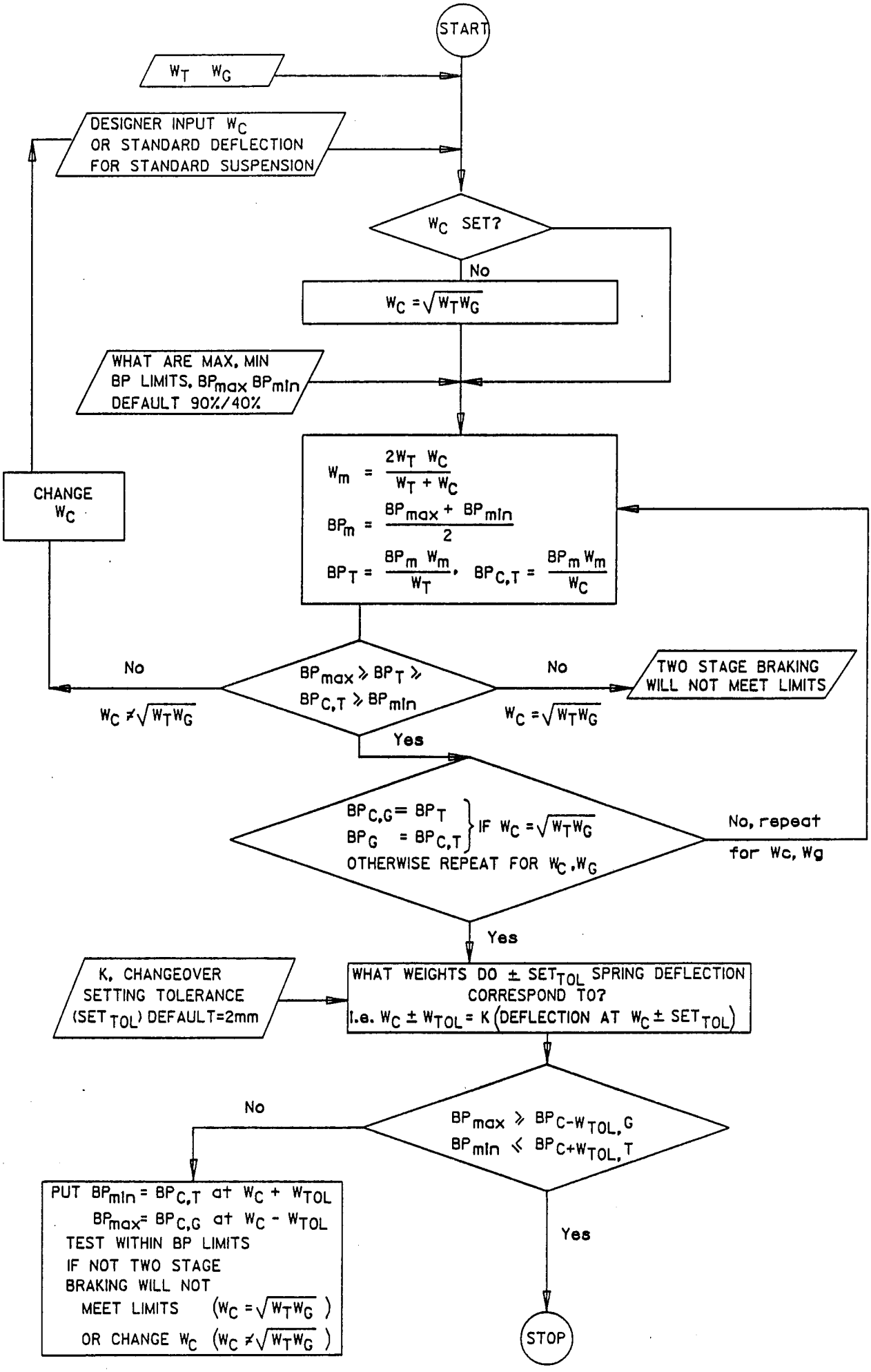


Figure 14. Wagon Brake Performance Strategies

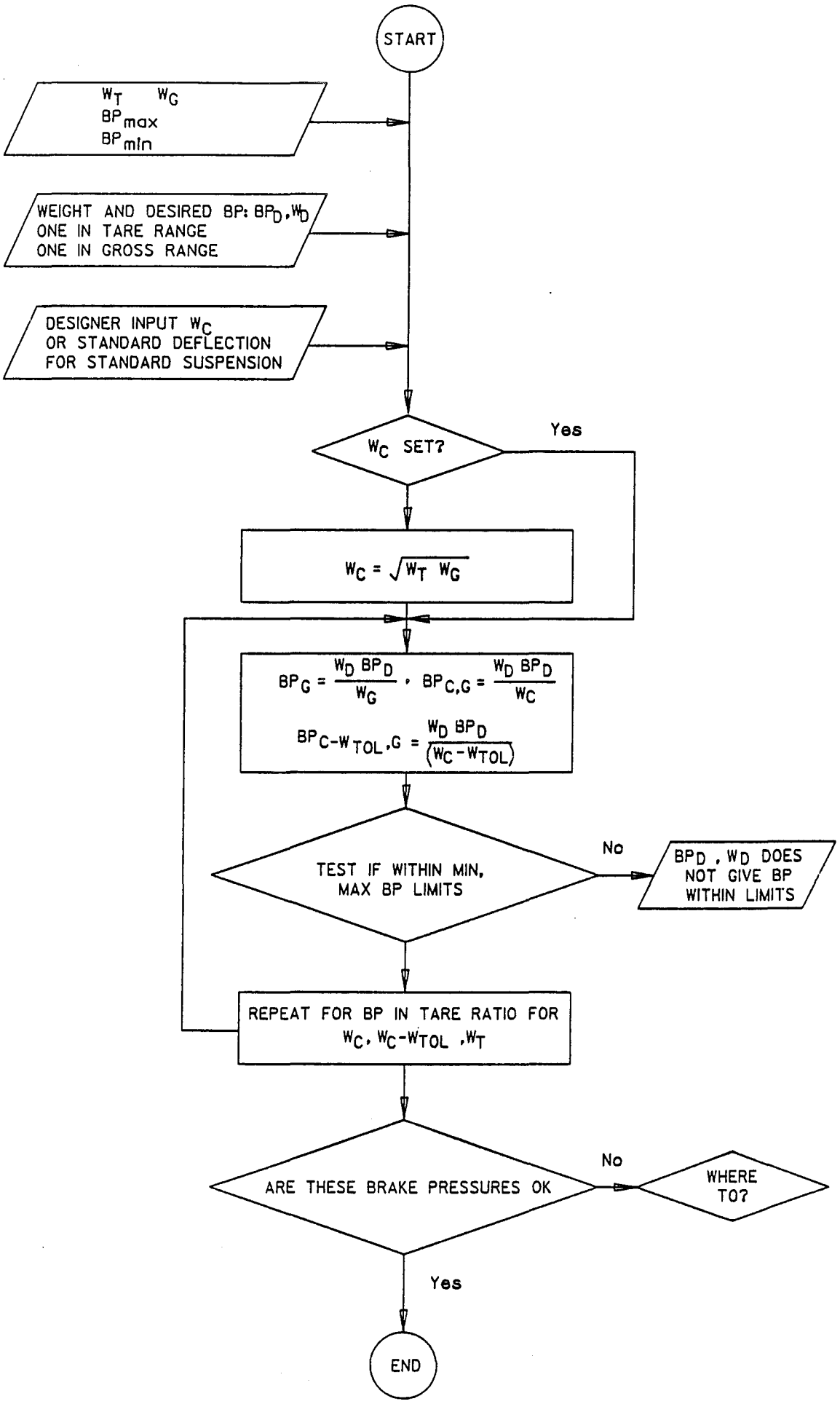


Figure 14. Wagon Brake Performance Strategies (cont'd.)

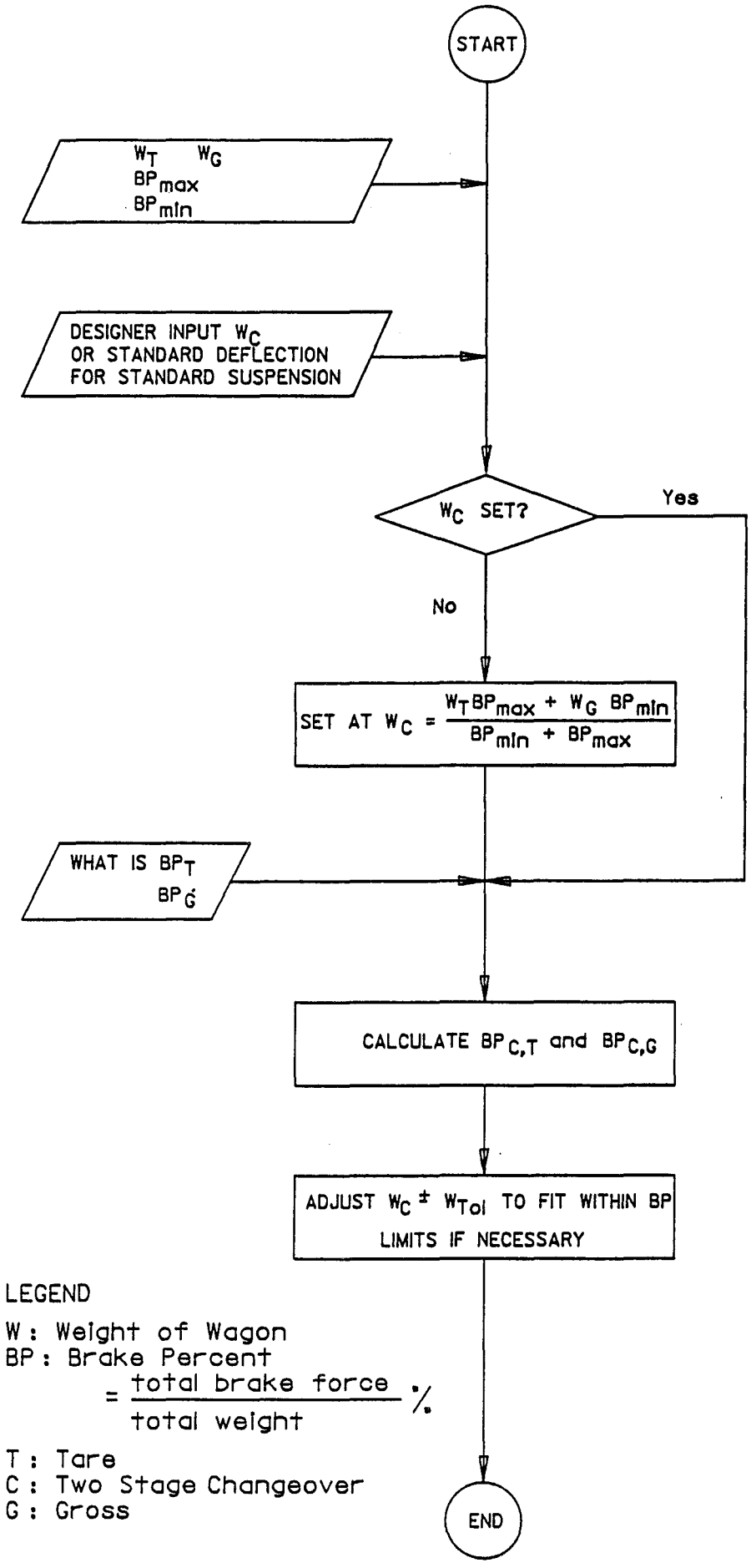


Figure 14. Wagon Brake Performance Strategies (cont'd.)

load. There are two main types; two stage (or empty/load) and variable load. Again they are proprietary products that work by adjusting fulcrum points in the rigging, the pressure in the brake cylinder or the brake cylinder output force. Within NZR automatic devices have been used most frequently and the variable load devices have been restricted to trial usage. Although wheelslide protection equipment is available it has not been used on freight wagons within NZR.

These load compensating devices are used when the ratio of the total brake force to the total weight of the wagon rail load (as a percentage this is called "brake percent") exceeds values determined from experience (Australia and New Zealand Railways, 1977; Assoc. of American Railroads, 1978b; NZR CMEO Design Office : Ksx, Nh, Nk, Pk, Ucg, Usg, Zm Design Notes). These limits indirectly represent underbraking (or excessive stopping distance) and overbraking (wheelsliding or excessive heat generation). Sometimes stopping distance and wheelsliding have been used directly as performance measures (NZR CMEO Design Office : Ksx, Nh, Nk, Pk, Ucg, Usg, Zm Design Notes).

As the vehicle has its load increased the brake percent drops for a constant braking force system and if the ratio of gross weight to tare weight is large these performance limits may be exceeded and the Australia and New Zealand Railways (1977), for example, specifies a maximum gross weight to tare weight ratio for constant braking force systems. Figure 14 shows some brake percent performance strategies.

Although these constraints force the use of load compensating equipment, the wagon designer has sometimes decided to use this equipment where it was not required because of benefits in performance. Compatible performance in trains is important as excessive

difference in braking performance among wagons can lead to longitudinal train shocks, wear and overheating defects in the brake drums.

On wagons with a high probability of load imbalance from one end to the other (for example, a bogie container wagon) or on a particularly long wagon, two change over valves or two variable load devices (one per wagon end) and/or two independent brake rigging systems each with its own brake cylinder have been used. Change over valves are valves mounted beside the suspension that measure the suspension deflection in order to activate the device that changes the empty loaded fulcrum pivots.

(6) Handbrake Equipment

Similar considerations apply to the handbrake equipment. The predominant type used is that of a ratchet foot lever at the end of the wagon, although handwheel geared types operable from the side of the vehicle and proprietary vertical handwheel types operated from a step at the end of the wagon have been used. Most Design Office handbrake designs deliver a force to the air brake rigging but one disc braked wagon design (Nk) has separate tread brake shoes and rigging.

Again empirically determined brake percent values are used as handbrake performance limits (Australia and New Zealand Railways, 1977; Assoc. of American Railroads, 1978b). Occasionally wheelsliding and a maximum grade on which the handbrake must hold a stationary wagon have been performance measures. Compatibility of performance is important for operating staff and present marshalling yard practice involves the use of the handbrake to slow down a free running wagon or rake of wagons. Another important constraint with the ratchet lever is that the

maximum travel (due to brakeblock clearances, wear, etc.) does not exceed the device's capacity and that the lever, which is located underneath the wagon, in the full travel position does not interfere with the track, particularly in vertical curves.

(7) Brake Blocks

The brake block characteristics (particularly the coefficient of friction) influences the design of the other brake system components. High friction (relative to metal blocks) composition brake blocks reduce the brake block load required, enabling smaller diameter brake cylinders and/or reduced rigging leverage to be employed. For high coefficient of friction blocks the brake percentage limits are correspondingly lower, but an "equivalent" cast iron block brake percentage can be calculated. The use of high friction blocks influences the choice between clasp (two opposing brake blocks per wheel) and single brake block rigging. Single block rigging offers less weight and fewer parts to fail which in turn increases availability and reduces derailment hazard.

The performance characteristics of brake blocks have generally been determined by tests on dynamometers and by track tests. Galton's classical work on cast iron blocks in the 1870's and other early experimental works are discussed in Koffman (1948). Test results and empirical equations are presented relating the block's coefficient of friction to velocity, in particular, but also to application pressure, block hardness, duration of application and temperature, and to the shape of the block and its flexibility. The block hardness and total work done had a less dramatic effect on block wear than application

pressure and speed (collectively power dissipation), the results suggesting acceptable limits. The tyre or wheel steel acceptable power dissipation limits must also be considered.

More recently Manos and others (1981) have published results of stop, drag and holding track tests that aimed to determine the influence of moisture, speed and temperature on the coefficient of friction of selected "composition", high phosphorous cast iron, and cast iron tread brake blocks. The measured vehicle windage and rolling resistance loss values, which were four to ten percent of braking forces, were in good agreement with an empirical equation (the "Davis" equation). Lagedrost and others (1979) have presented thermal property measurements of cast iron and composition block materials at various temperatures.

Other considerations in the selection of friction materials include noise and jerkiness of braking, sparking and block ignition, block cracking and material breakup tendencies, and block weight, and block handling requirements. The deleterious effects of wear debris are a major factor as the blocks may contain substances such as asbestos and quartz; the hot debris can cause fires; it may soil vehicles and trackside buildings; and it may cause the malfunction of electrical equipment. Waldron (1979) has surveyed the factors influencing the choice of friction braking materials in tread and conventional disc brakes.

McGuire (1979) has attempted to predict the wear of composition brake blocks but found only that relative wear rates had reasonable agreement with test values. McGuire concluded wear was dependent on temperature and rubbing velocity.

Various organisations' standards specify performance and physical

characteristics of brake blocks, block heads, shoe spear, and so on. Proprietary brake blocks, heads, and spears meeting these standards and existing designs are available to the NZR wagon designer.

3. RIGGING ARRANGEMENT DESIGN

The total lever ratio relates the brake block force to cylinder output. Depending upon the accuracy desired or the information known at the time, losses due to springs, slack adjustors, friction and so on have been included. The wagon structural layout can have a significant effect on rigging layout. For example, a hopper wagon with bottom doors precludes the use of a central cylinder. Layouts and parts from earlier designs have often been used again,bogie brake designs in particular are used repeatedly (with their respective bogie designs). Figure 15 shows some common wheel and cylinder lever arrangements. The total lever ratio and rigging arrangement must be reconciled to give a rigging arrangement that is light, efficient, cheap, subject to minimum wear, and that at maximum movement (that is, applied to worn wheels with a worn set of brake blocks) does not interfere with any part of the wagon or its equipment (which in turn may be capable of movement, for example, hopper doors or bogies or the brake system itself).

High rigging efficiency can reduce air requirements, and reduce the size of parts. It is important that the minimum brake pipe pressure reduction overcomes the rigging losses and gives some vehicle retardation. The main causes of low efficiency are cylinder, spring and slack adjuster losses, and friction and lever obliquity (which introduces force obliquity and may cause friction losses such as push rod binding against cylinder-sleeve). Carman (1971) has presented some

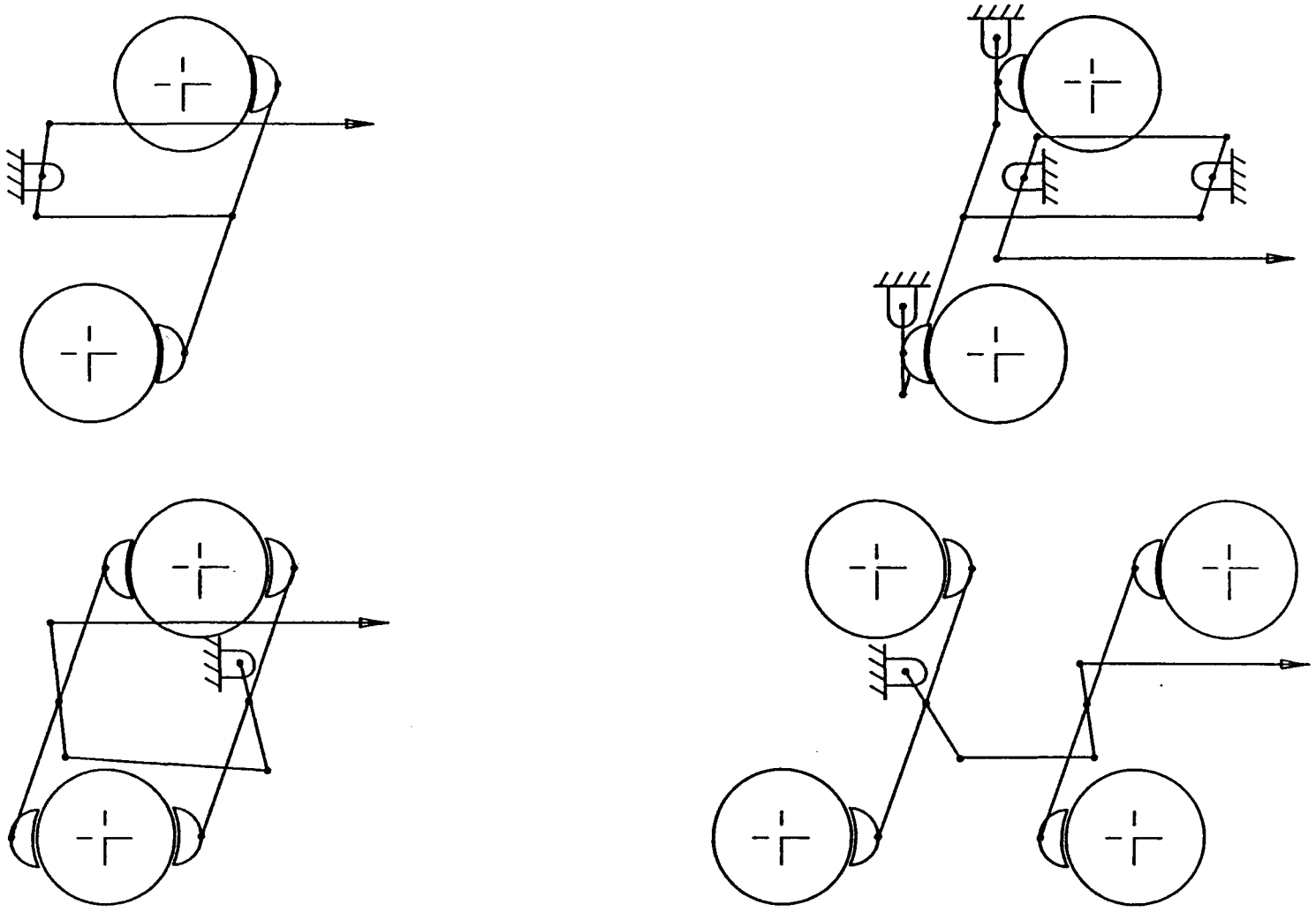


Figure 15. Wagon Brake Rigging Wheel and Cylinder Arrangements

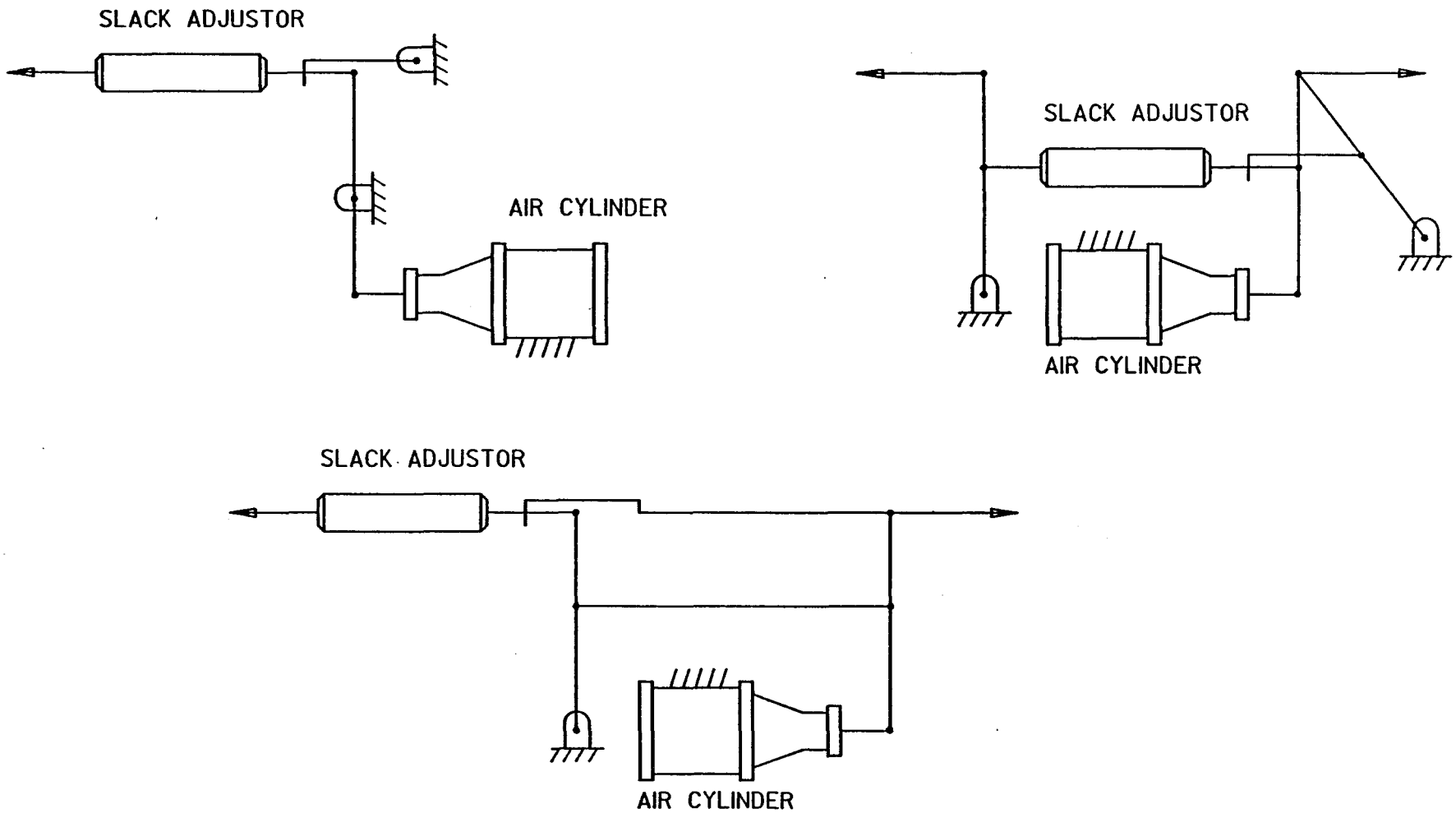


Figure 15. Wagon Brake Rigging Wheel and Cylinder Arrangements (cont'd.)

results and empirical equations for the comparison of static and dynamic rigging efficiencies. The losses were significant but showed considerable variation.

The rigging is often self equalising, balancing the brake forces one end to the other and between brake blocks acting on a wheelset. The rigging must also remain in stable equilibrium and not unduly influence suspension and wheelset guidance functions. Vibration of the rigging produces wear of the pins and links, and therefore, the rigging should be readily accessible for inspection and replacement purposes. Because of vibration spring washers, self-locking nuts, and other means of preventing loosening of bolted joints are used. Broadbent (1956) has studied the forces and movements in clasp brake rigging and offered an explanation of "brake chatter" or oscillation.

As information changes or as it becomes available during the design process the detailed layout can be modified or developed. From the geometry of the levers, brake beams, blocks and wagon structure, decisions can be made on the pullrod lengths (and their adjustment range) and the location of fulcrum point brackets (and their design if necessary).

In order to set up piston travel and the brake rigging in manufacture or repair, the piston false travel (that is, the difference between specified total piston travel and the piston travel required to take up the brake block clearance and rigging slack) is often calculated, see figure 16. The brake cylinder travel plus the slack adjuster take-up must be sufficient to allow for the full range of wear of a set of brake blocks.

Short stroke brake cylinder	N	Y	N
Long stroke brake cylinder	Y	N	Y
Two stage braking	Y	Y	N
S.A.B. regulator	Y	Y	Y
Single stage braking	N	N	Y
Travel = 140mm in "Load"	X		
Travel = 100mm in "Load"		X	
Travel = 125-130mm			X

Figure 16. Wagon Brake Piston Travel

4. PNEUMATIC ARRANGEMENT DESIGN

As with the rigging layout the pneumatic layout design must take into account interference with suspension or other equipment movement; piping through or around the wagon structure; weight and weight distribution; proneness to damage; previous designs or standard parts; ease of removal of defective subassemblies; ease of inspection; and so on. Air flow friction losses affect brake pipe pressure reduction time and restoration time and consequently train stopping distances and recharge times. These losses may be reduced by minimising the number and severity of bends, the length of piping and bore roughness (which in turn may influence first cost). Other failures can result from foreign matter in the pneumatic gear, fracture of pipes at stress concentrations, inadequate anchorage of pipes, and leakage of seals at joints. Leakage can result in excessive brake release times, premature brake release, differences in wagon cylinder pressures, no empty/load change over, and other malfunctions. The manuals of standards and recommended practices contain bending radii, pipe sizes, types of pipe clamps and their spacings, moisture trap specifications, types of unions and joints. Figure 17 contains an example of a parameterised four wheel wagon brake pipe layout.

When the branch pipe lengths were not in the normal range of lengths or when novel pneumatic designs were used, Boyle's Gas Law was applied to adjust pipe lengths or reservoir volumes, or to check brake cylinder or reservoir pressures. Calculations of this sort required information such as brake pipe pressure, pneumatic layout, triple valve function, variable load valve (or other specialised control valve)

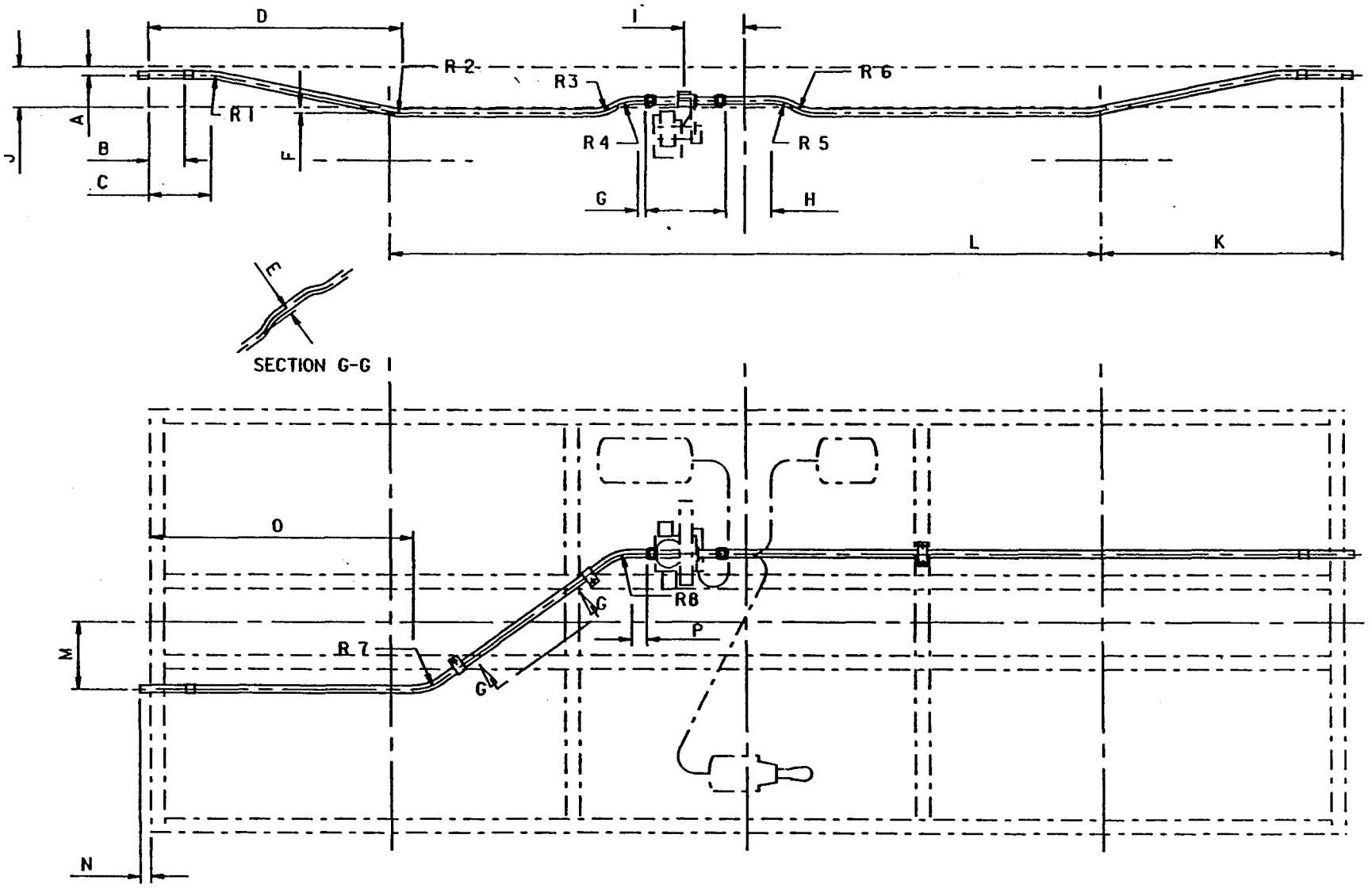


Figure 17. Parameterised Four Wheel Wagon Brake Pipe Layout

function.

Detailing the pneumatic system involves selection from the available variations of the pneumatic devices, for example, triple valves should match the brake cylinder and auxiliary reservoir on the vehicle. Figure 18 illustrates the use of WF2 type triple valves.

5. RIGGING COMPONENT DESIGN

Once the forces in the rigging have been determined (from the cylinder or brake block forces and rigging leverages), the levers, pins, links, and so on, can be designed for strength and endurance. Although the components are subject to repeated loading and unloading, calculations predicting fatigue life are not generally performed, but stresses resulting from a maximum static load are compared with a suitable design stress and design factor. The design manuals (Australia and New Zealand Railways, 1977; Assoc. of American Railroads, 1978b) have design loads, stress limits, minimum dimensions, and standard dimensions for various components

Pins may be selected from the range of standard pins availiable or from existing designs on the basis of the duty required and the nominal capacity of the pin. Alternatively they may be selected as part of a standard or previously designed subassembly. Or the diameter may be calculated by using the design stress and the shear load, and compared with that of existing pins. The same pin diameter is often used at both ends of a pull/push rod. Standard bush dimensions are obtained from Y42102 or if unbushed a nominal clearance hole is specified. The length of the pin depends upon lever and/or rod thickness and clearance,

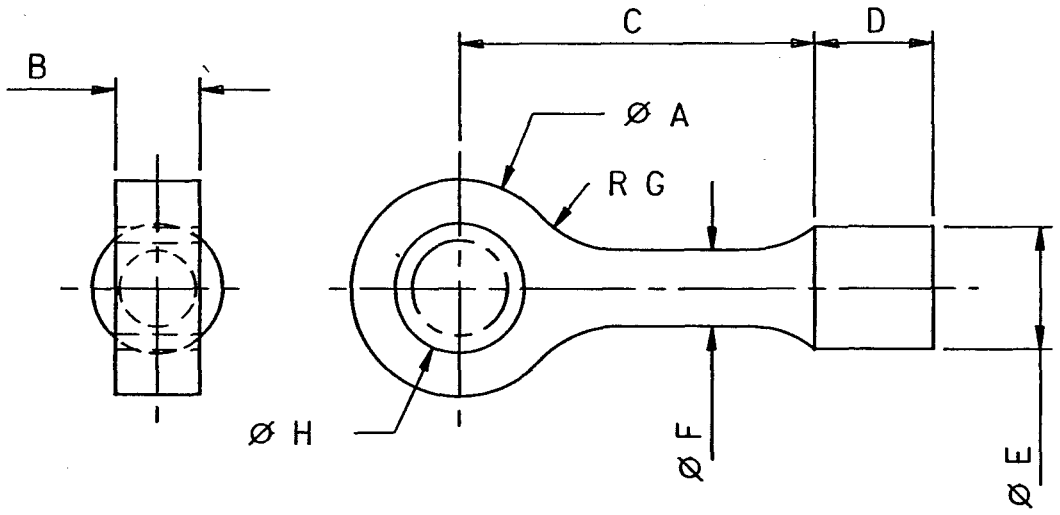
Four wheel wagon	Y	N	N	N	N
Bogie wagon	N	Y	Y	Y	Y
203mm short stroke brake cylinder	Y	N	N	N	N
152mm long stroke brake cylinder	Y	N	N	N	N
254mm long stroke brake cylinder	N	Y	N	N	N
2 No. 203mm short stroke brake cylinders	N	Y	N	N	N
203mm long stroke brake cylinder	N	N	Y	N	N
254mm short stroke brake cylinder	N	N	Y	N	N
305mm long stroke brake cylinder	N	N	N	Y	N
WF-ELS relay valve	N	N	N	N	Y
A.C.S. WF2 valve	X				
A.C.L. WF2 valve		X			
A.C.T. WF2 valve			X		
A.C.X. WF2 valve				X	
W. relay WF2 valve					X

Figure 18. Wagon WF2 Triple Application

sufficient thickness is given to withstand bearing stresses. Similar considerations apply to lever and pullrod design. The lever section modulus depends on its shape and size, the bending moment on the applied forces and lever lengths, whereas the push rod is checked for buckling, the pinned joint for bearing, shear, tension stresses and the eccentric component for combined bending and direct action. Brackets, stops, chains, shackles, brake beams, welded and bolted connections, and shafts are other components that are sometimes checked for strength. Figure 19 illustrates some parameterised geometry of brake rigging parts.

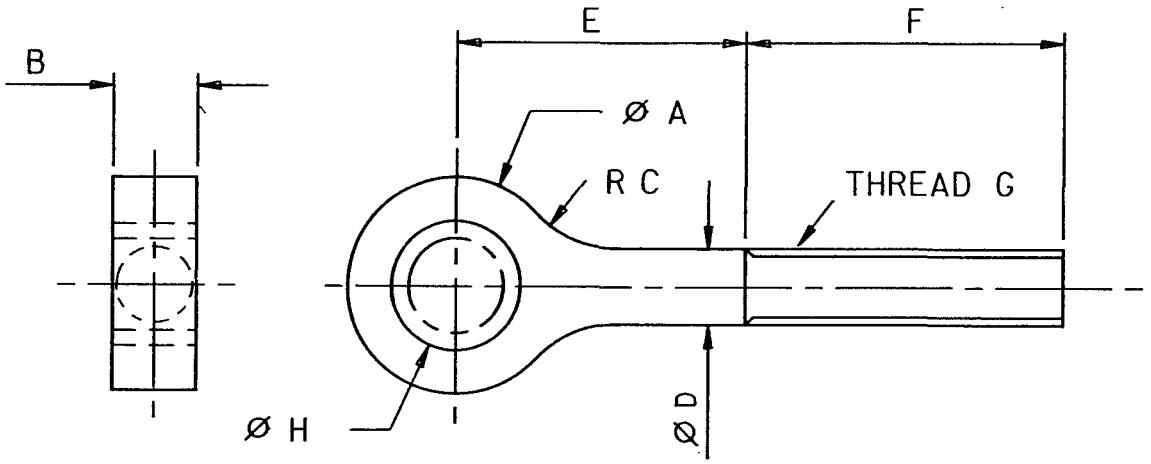
6. VERIFICATION OF THE BRAKE ARRANGEMENT DESIGN

Whereas in the early stages of the brake design the strategy is to determine the equipment that should meet the various criteria such as stopping distance or brake percent, later, when information on the details is available, more comprehensive checks are made to ensure that the brake system as a whole meets the design criteria. The Design Office has produced a brake performance program that runs on their HP desk top calculator. The program reads in speed, leverage, coefficient of friction, and so on, and outputs brake block force, brake percent, stopping distance, retarding force, and so on. At this stage it is possible to include in the brake performance calculations the effects of rolling resistance; grade; rotational inertia; spring and friction losses; coefficient of friction variations with velocity, pressure, and so on; and cylinder pressure variation with time. However, if simplified mechanics are used it is often possible to rearrange expressions to relate one quantity in terms of others. For example, the deceleration



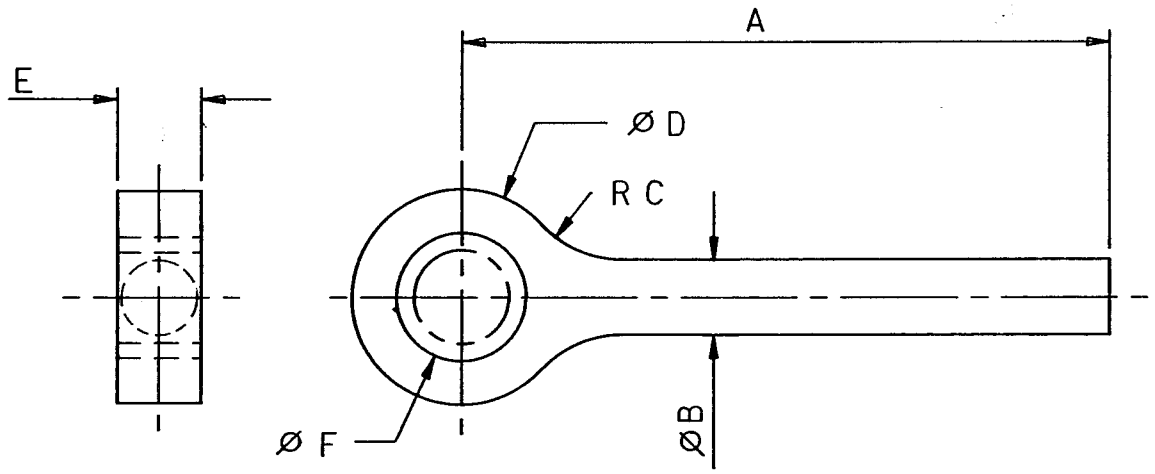
IDENTIFICATION	NAME	A	B	C	D	E	F	G	H	BUSH
A on 27149/2	Pullrod end	2 "	1 "	3"	1 1/4"	1 1/4"	7/8"	1 1/8"	1 1/4"	Yes
B on 27149/2	Pullrod end	2 "	1 "	3"	1 1/4"	1 1/4"	7/8"	1 1/8"	29/32"	No
C on 27149/2	Pullrod end	2 "	1 "	76	1 1/4"	1 1/4"	7/8"	1 1/8"	34	Yes
28796/3	Pullrod end	57	25	440	40	40	22	28	23	No
A on 28074/1	Pullrod end	64	25	76	32	31	22	30	34	Yes
B on 28074/1	Pullrod end	64	25	76	32	31	22	30	23	No

Figure 19. Parameterised Geometry of Wagon Brake Rigging Parts



IDENTIFICATION	NAME	A	B	C	D	E	F	G	H	BUSH
A on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	4¼"	9½"	1" W	1¼"	Yes
B on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	3'6"	5"	1" W	29/32"	No
C on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	350	240	M24	34	Yes
D on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	1120	130	M24	34	Yes
E on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	1630	130	M24	34	Yes
F on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	2520	130	M24	23	No
G on 27149/3	Pullrod	2¼"	1"	1⅛"	⅞"	1280	130	M24	34	Yes
A on 28074/4	H.B. pullrod	64	25	30	22	110	240	1" W	34	Yes
B on 28074/4	H.B. pullrod	64	25	30	22	1100	130	1" W	23	No
C on 28074/4	H.B. pullrod	64	25	30	22	780	130	1" W	23	No
28074/6	H.B. fulcrum pullrod	64	25	30	22	175	75	⅞" W	22	No
A on 28808/5	Pullrod screwed	57	25	32	22	486	127	M24	23	No
B on 28808/5	Pullrod screwed	57	25	32	22	303	127	M24	23	No
C on 28808/5	Pullrod screwed	57	25	32	22	573	127	M24	23	No
D on 28808/5	Pullrod screwed	57	25	32	22	193	127	M24	23	No

Figure 19. Parameterised Geometry of Wagon Brake Rigging Parts (cont'd.)



IDENTIFICATION	NAME	A	B	C	D	E	F	BUSH
29481/2	Pullrod (SA end)	165	22	30	64	25	38	Yes
A on 28836/1	Fulcrum rod empty	732	22	30	64	28	38	Yes
B on 28836/1	Fulcrum rod empty	754	22	30	64	28	38	Yes
28836/2	Pullrod (ASA end)	265	22	30	64	25	38	Yes
A on 28124/8	Fulcrum rod (tare)	782	25	38	75	28.6	42	Yes
B on 28124/8	Fulcrum rod (tare)	763	25	38	75	28.6	42	Yes
C on 28124/8	Fulcrum rod (tare)	749	25	38	75	28.6	42	Yes
D on 28124/8	Fulcrum rod (tare)	757	25	38	75	28.6	42	Yes

Figure 19. Parameterised Geometry of Wagon Brake Rigging Parts (cont'd.)

rate may be calculated in terms of retarding force and weight; or in terms of speed and stopping distance, and the adhesion demand (friction demanded between wheel and rail) may be calculated in terms of retarding force and weight; in terms of speed and stopping distance; in terms of deceleration; or in terms of brake ratio and mean coefficient of friction. The quantities and expressions used appear to have depended on the information available at the time and on the designer's preferences.

Howard (1983) has presented the details of an interactive procedure for designing the leverage of air brake rigging. The program, which is based on design reports, performs calculations, plots brake percent graphs, presents sketches of existing parts and checks empirical constraints. Through its question and answer type of dialogue with the user the program offers choices of methods and actions at each design decision stage.

7. WHEEL/RAIL ADHESION

The coefficient of adhesion is the ratio of tangential force to normal force at the wheel/rail interface. The limiting adhesion is the adhesion obtained just before sliding occurs.

Knowledge of adhesion is important as it is used in calculations aimed at preventing wheel damage caused by excessive braking and also in vehicle dynamics studies (stability and steering in particular). A review of the earlier work on both locomotive driving adhesion and wagon braking adhesion is given by Koffman (1948). More recently Logston and

Itami (1980) have presented results from investigations to determine the influence of rail condition, speed, track curvature, and axle load on adhesion and creep. Kumar (1980) has also reported on experimental creep studies.

Low adhesion is usually associated with rain or high humidity when rail wear debris, rail rust, industrial pollution, autumn leaf debris, coal dust, contamination from track mounted flange lubricators, or other contaminants mix with a critical amount of water to form a surface film with low shear resistance that is not readily squeezed from between the rail and wheel (Beagley, 1976 ; Barwell and Woolacott, 1963; Pritchard, 1979).

Pritchard (1979) also reported on the systematic measurements of braking adhesion carried out by British Railway's tribometer train. From the individual measurements, the means and standard deviations were determined (89% of the "runs" were found to form a Gaussian distribution) and thus probabilities of wheelslide or risk of slip with a certain adhesion demand were estimated. He compared the adhesion demand of various railway administrations and presented the results of small scale tests which confirm that some composition blocks reduce adhesion significantly, whereas cast iron blocks improve wheel/rail adhesion.

Another major influence on wheelslip is reduction in wheel load. Tack (1951) derived some equations from basic statics and dynamics to describe the effect on wheel load of track irregularities, wheel out of balance, eccentric wheels, and the transfer of weight distribution over the axles with braking. Vehicle dynamics is discussed in chapter V. But at present the full effect of speed on adhesion cannot be predicted for any given vehicle.

8. REFERENCES

ANON. (1979) Materials for Brake Discs of Rail Vehicles. Institution of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.189-194. (Institution of Mechanical Engineers Conf. publication 1979-11).

ASSOC. OF AMERICAN RAILROADS. MECHANICAL DIVISION (1978a) Manual of Standards and Recommended Practices: Section E - Part I: performance specification for single capacity freight brakes. Rev. ed. p. E-95 - E-102.

ASSOC. OF AMERICAN RAILROADS. MECHANICAL DIVISION (1978b) Manual of Standards and Recommended Practices: Section E - Part I : basic freight car brake design data. Rev.ed. p. E4 - E10.

AUSTRALIA AND NEW ZEALAND RAILWAYS. [1972] Brakes and Brake Equipment : general principles and fundamentals of design. p. E1 - E5.

BARWELL, F.T. and WOOLACOTT, R.G. (1963) The N.E.L. Contribution to Adhesion Studies. Proc. Institution of Mechanical Engineers, 178 (part 3E) : 145-160.

BEAGLEY, T.M. (1976) The Rheological Properties of Solid Rail Contaminants and Their Effect on Wheel/Rail Adhesion. Proc. Institution of Mechanical Engineers, 190:419-428.

BLAINE, D.G. and others. (1981) Train Brake and Track Capacity Requirements for the '80's. Journal of Engineering for Industry, Trans. ASME, 103 : 392-401.

BROADBENT, H.R. (1956) Forces on a Brake Block and Brake

Chatter. Proc. Institution of Mechanical Engineers, 170 : 993-1008.

CARMAN, R.W. (1971) Brake Rigging Efficiency of Railway Freight Cars. ASME Paper No.71-WA/RT-2.

HART, J.E. and GREJDA, F.J. (1972) Computer Simulation of Rail Vehicle Braking Performance. ASME Paper No.72-WA/RT-5.

HOWARD, A.A.B. (1983) Computer-aided Design of Lever Brake Systems. Christchurch, University of Canterbury. (Final year project report : B.E.: Mechanical Engineering).

KOFFMAN, J. (1948) Adhesion and Friction in Rail Traction. Journal of the Institution of Locomotive Engineers, 38 : 593-672.

KUMAR, S. (1980) Some Wheel-rail Interaction Factors Influencing Vehicle Dynamics. Rail International, 11(10) : 599-610.

LAGEDROST, J.F. and others. (1979) Thermal Property Measurements in Brake Shoe Materials. Institution of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.111-114. (Institution of Mechanical Engineers Conf. publication 1979-11).

LOGSTON, C.F. and ITAMI, G.S. (1980) Locomotive Friction-creep Studies. Journal of Engineering for Industry, Trans. ASME , 102 :275-281.

MCGUIRE, M. (1979) Predicting the Wear of Composition Brake Blocks. Institution of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p. 143-151. (Institution of Mechanical Engineers Conf. publication 1979-11).

MANOS, W.P. and others. (1981) Brake Shoe Performance Evaluation: Vol 1. Chicago, Illinois, AAR Technical Division. 140 p. (AAR Research and Test Dept. Report No. R-469).

NZR. CMEO Design Office. Ksx Wagon Brake Design Notes. [1976].

NZR. CMEO Design Office. Nh Wagon Design Notes. [1977].

NZR. CMEO Design Office. Nk Wagon Design Notes. [1980].

NZR. CMEO Design Office. Pk Wagon Design Notes. [1980].

NZR. CMEO Design Office. Ucg Wagon Design Notes. [1980].

NZR. CMEO Design Office. Usg Wagon Brake Design Notes. [1979].

NZR. CMEO Design Office. Zm Wagon Design Notes. [1974].

PRITCHARD, C. (1979) Brakes and Wheel/rail Adhesion. Institution of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.19-26. (Institution of Mechanical Engineers Conf. publication 1979-11).

SHUTE, B.W. and others. (1979) The Effect of Leakage Distribution on Brake Pipe Gradient and Brake Pipe Flow. ASME Paper No. 79-WA/RT-16.

TACK, C.E. (1951) Development and Testing of Brakes for High-speed Railroad Equipment. Trans. ASME, 73 :167-182.

TICKLE, C.J.F. (1979) The Rail Vehicle Disc Brake : principle and practice. Institution of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.167-181. (Institution of Mechanical Engineers Conf. publication 1979-11).

VINK, P. (1981) Derailment Statistics for Years 1979 and 1980. NZR Way and Works Branch Research Report 192.

WALDRON, G.W.J. (1979) The Choice of Materials in Friction

Braking. Institution of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.125-135. (Institution of Mechanical Engineers Conf. publication 1979-11).

CHAPTER VII

WAGON WHEELSET DESIGN

1. DEFINITION, FUNCTION, AND PERFORMANCE

The wheelset is the assembly that guides and supports the vehicle's suspension along the railway track and usually with freight wagons transmits the braking forces to the rail.

For the purposes of this thesis the wheelset consists of the axle and the two wheels attached and the two bearings and their immediate housing - the axlebox. Discs for braking may also be part of the wheelset.

Sudden complete failure of this assembly results in damage to other railway components and occasionally loss of human life. Safety and reliability are important criteria in its design. For the four years 1977 to 1980 New Zealand Railways had a total of 6.5 ("0.5" because there were other contributing factors to a derailment) derailments attributed to wheel defects, 11 to tyre defects and 10 to axle defects totalling 34% of all derailments caused by vehicle defects (Vink, 1981).

The development of wheelsets is driven by commercial pressures to produce wheelsets that are cheaper to produce and maintain, and can withstand heavier axle loads, increased thermal abuse (where the wheelset must act as brake drum) and have extended mileage to wear-out. In a railway system with tight tunnel clearances (such as in NZR)

decreasing the height of the wheelset assembly is also important. Wheelset mass plays a vital role in cost effective operation. "With unit train consists [train composition] currently in use each additional 15-20kg of wheel mass is equivalent to one less ore car [mineral ore wagon] in the train consist." (Long, 1978, p.1). The unsprung mass of the wheelset is a factor in vehicle stability and dynamic loads on the track.

Current NZR wheelset designs do not permit rotation of the wheels on the axle, and the distance between the two wheels must be maintained. Excessive difference in tread diameters of the wheels on the same axle is also detrimental to performance.

2. THE WHEEL

(1) Function of the Wheel

The rolling of flanged steel wheels along steel rails is an essential process of railways, conceptually unchanged for nearly 200 years. It makes possible mass transport in self guided vehicles under conditions of very low tractive resistance. The primary functions of the wheel, then, are:

(a) Support of the weight of the vehicle and its lading.

(b) Guide the vehicle in the track.

In the case of vehicles that use the wheelset as a component in the brake system it must also transmit braking forces and frequently acts as a brake drum.

(2) Wheel Failure Modes

Thus most freight wagon wheels must withstand not only the dynamic stresses that result from vertical and lateral rolling loadings but also thermal effects that result from the friction created during braking. This duty is such that on current wheel designs defects occasionally develop, these defects are usually detected by inspection and the wheel is removed from service before it is worn out. Infrequently complete failures occur resulting in derailments.

The principal types of problems that occur in service are (Ryan and Hundy, 1960):

(a) Wear. Wear of the tread and flange (the running surfaces of the wheel) depends on a number of factors including the material of rail and wheel, lubrication and climatic conditions, the design of the vehicle, operating speeds, and characteristics of the track. Flange wear is particularly important as a thin flange can lead to the removal of many times the flange wear from the tread to re-form the flange. If hollowness of the tread develops too far vehicle dynamic performance may deteriorate.

The limits of wear, for example, flange height and tread guttering, flat size, sharpness of flange, thinness of flange and tyre/rim thickness, are specified for existing wheel designs in the Mechanical Branch Codes and drawings.

(b) Thermal Effects. Temperature gradients that develop in the wheel are dependent upon the amount of energy dissipated and the rate at which is is dissipated during each brake application. Temperatures at the tread surface can easily reach 800°C in local hot spots a few centimeters in diameter - twice the bulk tread temperature. Cracks in

the tread have been associated with the microstructure changes in the material (the diffusion of carbon into the tread and the formation and tempering of martensite (Wandrisco and Dewez, 1960)) and with cyclic thermal strain (Hewitt and Musiol, 1979). There is some evidence that these cracks do not progress beyond the local heat-affected zone but are continually worn away. However frequent brake applications may cause crack growth particularly if the crack initiation site is relatively free from wear and has a thinner compressive work hardened layer such as flange tip or front edge of tread (Wandrisco and Dewez, 1960).

Residual stress changes may also occur in the tread when the expanded hot surface is restrained by the cooler metal beneath it, to such as extent that the surface material is plastically deformed.

When the adhesive force between wheel and rail is less than the braking force the wheels will slide resulting in rapid wear and development of a local hot spot. The abrupt cooling of the flat spot leads to local microstructure changes, and this harder, more brittle disc may in turn be spalled out of the tread.

In other service conditions such as in drag braking where brakes are used to check a train down a long gradient the whole rim may expand causing plastic deformation or fracture in the web of the wheel (particularly at the boss and rim fillets) and high residual tensile stresses in the rim upon cooling. High carbon wheels have developed sudden cracks (with a brittle fracture surface) as a result of "drag" brake applications and have even exploded as they cooled (Ryan and Hundy, 1960; Wandrisco and Dewez, 1960; Wetenkamp and others, 1950). Repeated applications can again lead to fatigue failure.

Another common thermally induced failure is the expansion of a tyre on its centre allowing its rotation around the centre. In extreme cases, perhaps caused by the failure of the brakes to release, the wheel may be loosened on the axle.

(c) Fatigue. Most fatigue cracks are found to start from a stress raiser of some sort, for example, thermal cracks, rim stamping marks and inclusions.

Shelling is characterised by expulsion of pieces of metal from the tread surface and is essentially a fatigue failure caused by rolling loads (Wandrisco and Dewez, 1960). When a piece of tread shells out between thermal cracks the process is called spalling.

(d) Brittle Failure. Brittle failure is usually associated with a stress raiser and high tensile stress. This tensile stress may be residual as in the case of oil quenched wheels or may occur in service due to thermal stresses or impacts with a bad rail joint or a crossing frog.

(e) Corrosion and Corrosion Fatigue. Atmospheric corrosion appears to have an accelerating effect on the loss of metal at the running surfaces of the wheel (Dearden, 1960). Fretting corrosion can occur at the wheel/axle interface and tyre/wheel centre interface possibly leading to corrosion fatigue.

(f) Dimensional Instability. Thermal or mechanical loads can permanently or elastically change the shape of the wheel. Lateral wheel deflection can adversely affect wheelset gauge; changes in wheel tread conicity can result from rim twisting; and, as discussed earlier, press fits can be loosened.

(3) Wheel Types

Economic factors, such as first cost and estimated life, influence the decisions on the type of wheel to adopt - one wear, multi-wear solid wheels; or built-up wheels. (A built-up wheel has a tyre fitted to a steel disc centre usually fastened with a press-fit and retaining ring.) If the service involves heavy wear the built-up wheel may offer economic advantage particularly if use is made of expensive alloy steels. But other factors must be considered such as weight, risk of corrosion, circumferential tensile stress in tyres increasing crack propagation rates, repair facilities and maintenance required, scrap values, and so on.

In the past spoked wheels have been used, but they permit flexing of the rim (which in built-up wheels leads to fretting corrosion) and produce irregular stress patterns.

Despite the wheel being a potentially large heat absorbing and dissipating mass, and the wearing action of brake blocks being good for adhesion and removal of superficial tread cracks, separate specialised discs are sometimes used for braking the vehicle. Disc braking eliminates thermal problems from the wheel. If the disc is attached to the web, or is an integral part of the web, as opposed to being a separate disc attached to the axle, it imposes constraints on geometry and on other performance requirements of the wheel web.

(4) Running Surface Design

(a) Contact Stresses. A prerequisite to the prediction of shelling and plastic flow in the wheel running surfaces is accurate modelling of contact stresses.

Traditionally the working rule was:

$$P/D = \text{constant},$$

which related the safe working axle load P to the diameter of the wheels D (Ryan and Hundy, 1960). Research results in the field of contact stresses have been surveyed by Kalker (1979) and Johnson (1982). Most of the work uses the Hertz theory of contact, extending it to investigate rough surfaces and friction in rolling contact. In recent research (Paul and Hashemi, 1981; Kalker, 1979) numerical methods have been used to determine the contact area and pressure that arises when two elastic bodies with closely conforming frictionless surfaces of quite general geometries are pressed together. It is clear from Hertzian theory that plastic flow in the wheel tread occurs in normal service and Kumar (1980), for instance, has experimentally investigated the growth in contact area, plastic flow and wear with rolling cycles and various loads.

Jenkins and others (1974) studied wheel/rail vertical dynamic forces caused by various phenomena and suggested limits on wheel diameter, wheel static load, and vehicle speed for acceptable forces and stresses. Radford (1977), Frederick (1978), and Butcher and Horn (1978) applied this work to local conditions and presented further developments.

(b) Wear. The outside part of the tread can have increased coning to prevent excessive hollowness and reduce the impact on points and splice rails as the outside part of the worn treads run over them. This is an example of how most features of the tread and flange are a result of past experience.

Recent work has enabled the theoretical prediction of the effect of running surface or tyre profiles on vehicle dynamic behaviour (see

chapt. 5). This work also provides better wheel-rail force and position histories enabling calculation of wear rates. Some success in theoretical modelling of wear has been achieved (Armstrong, 1981). The threshold between mild and severe wear can be predicted in terms of yield strength of the metal involved and coefficient of friction between the surfaces.

(c) Braking. The braking system has considerable influence on wheels and tyres. Brake loads must be distributed evenly over all wheels and over the surface of each wheel so as to avoid wheel slide, local overheating of the tread and grooving. A number of experimental studies have been performed to determine the effect of brake block material on wheel temperatures, formation of deposits, and weldspots, and so on (for example, see Wetenkamp and others, 1980).

(5) Stress and Fatigue Analysis of the Wheel Under Mechanical and Thermal Loads

Stressing due to mechanical and thermal loads is a basis for choosing between alternative wheel designs.

Experimental stress analysis has been used to investigate existing wheel designs with thermal loads and rail vertical and lateral loads (Wetenkamp and Kipp, 1978; Bruner and others, 1967; Rusin and others, 1979) and the effect of method of manufacture, carbon content of material, heat treatment and wheel geometry on the ability to withstand severe heating (Wetenkamp and others, 1950).

Finite element analysis has been used in a number of cases to predict elastic stresses due to thermal and static mechanical loads (Hopper and others, 1977; Long, 1978). Johnson and others (1977) used

an elastic-plastic, isotropic hardening, von Mises material model to study the residual stress changes in an existing wheel design due to drag braking, and Thomas and others (1982) used an elasto-plastic analytical technique based on the kinematic hardening model and von Mises yield criterion to study the effects of cyclic thermal loads on the fatigue life of various wheels. They assumed a load history and computed fatigue life using the Miner's linear cumulative damage rule.

The temperature distributions of the wheel during and after braking have also been obtained using a number of programs and methods including finite difference and finite element, although approximating formulas for temperature rises and thermal stresses are available (Newcomb, 1979).

(6) Material Selection

Selection of material is important in wheel and tyre design. Carbon content affects hardness thereby changing wear resistance and resistance to shelling. Increasing carbon content can make the wheel more susceptible to thermal cracking and reduce its fracture toughness.

Analytical methods of fracture mechanics can be used to relate critical defect size to both material toughness and stress state (Long, 1978). Hewitt and Musiol (1979) have reported on a comprehensive attempt to improve wheel tread materials with the aid of a mathematical model of hot spots that relate internal strain and temperature distributions, temperature histories in service, and fatigue life. They determined data for the material model experimentally from material behaviour under conditions of cyclic thermal and mechanical loading at various temperatures and under various atmospheric conditions. Park and

Stone (1981) have also reported on cyclic behaviour of wheel steels. Alloy steels and other materials have been tested with reduced or full scale wheels on dynamometers.

Drag braking can lead to detrimental changes in residual stress levels and tempering of the rim hardness profile. But countering this the rolling action generates a strain hardened layer on the tread surface. Long (1978) has made some measurements of this work hardened layer. Wheels with higher carbon and alloyed steels may have reduced thickness of this beneficial cold worked layer.

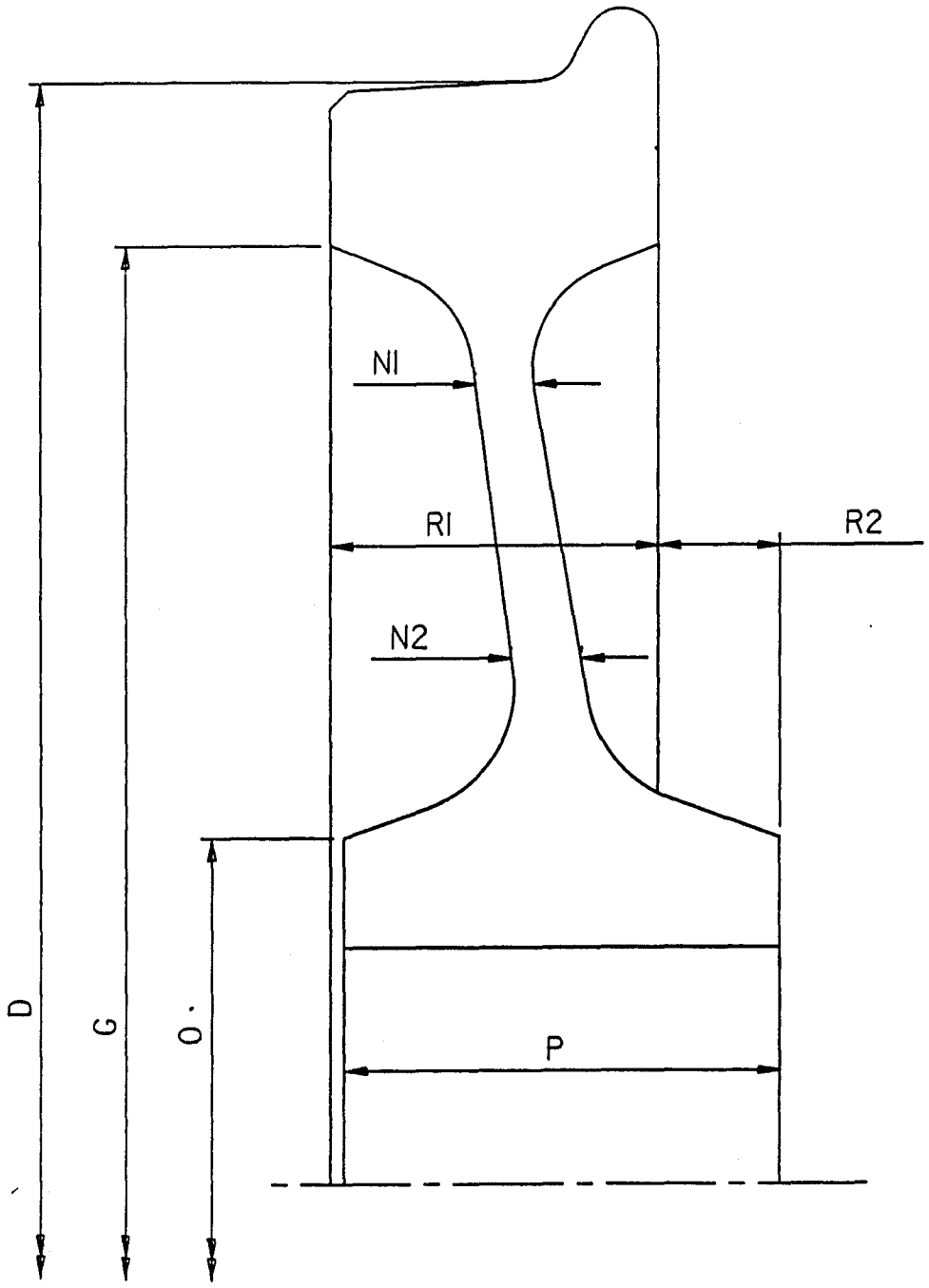
(7) Manufacturing Considerations

Concentricity, surface finish, groove radii, and placing of brand marks are machining specifications specified in order to eliminate stress raisers, and reduce the possibility of corrosion fatigue and dynamic load variations. Press fit assembly and disassembly techniques also affect the wheel design, for example, an oil injection groove may be provided to allow for disassembly by hydraulic wheelseat expansion. Process control and inspection to insure freedom from flaws are also important.

Wheels can be heat treated locally or as a whole to relieve undesirable residual stresses, or to introduce desirable residual stresses and properties. Long (1978) has described some measurements of residual stresses present in the wheel after manufacture.

(8) NZR Practice

Although mathematical modelling of the wheel has advanced considerably much of wheel design is based on a substantial body of experience (for example, see Ryan and Hundy, 1960).



IDENTIFICATION	D	G	N1	N2	O	P	R1	R2
W31507	768	654	19	25	241	152	114	43
W31471	840	690	22	30	270	180	127	64
OP3351	673	559	19	25	241	152	114	43

Figure 20. Wagon Wheel Geometries

For most wagons the Design Office selected a wheel from previous designs or from manufacturer's product ranges. National and international standards (for example ANZR, AAR, UIC, ISO) cover dimensions, tolerances and other essential properties of wheels and tyres although a user and a supplier may draw up their own specification. Figure 20 shows some wheel geometries which may be selected on the basis of axleload, brake system, wagon height restrictions, and so on.

3. THE AXLE

(1) Function and Performance

The axle must transmit the forces and moments between the wheels and bearings as it positively locates these components. The loads arise from numerous actions such as those occurring during wheel and rail interaction and those resulting from wind loads, coupler forces and the like on the wagon body. The axle may also have to transmit braking forces if, for example, there is an axle mounted disc brake. The axle must have enduring strength, and apart from the usual requirements of low production cost, and so on, low axle weight is also important because it is an unsprung mass.

(2) Axle Design in NZR

For many designs the designer selected the axle from a set of previous designs on the basis of axle load, suspension type and possibly wheel size. However changes in wheel diameter, height of centre of

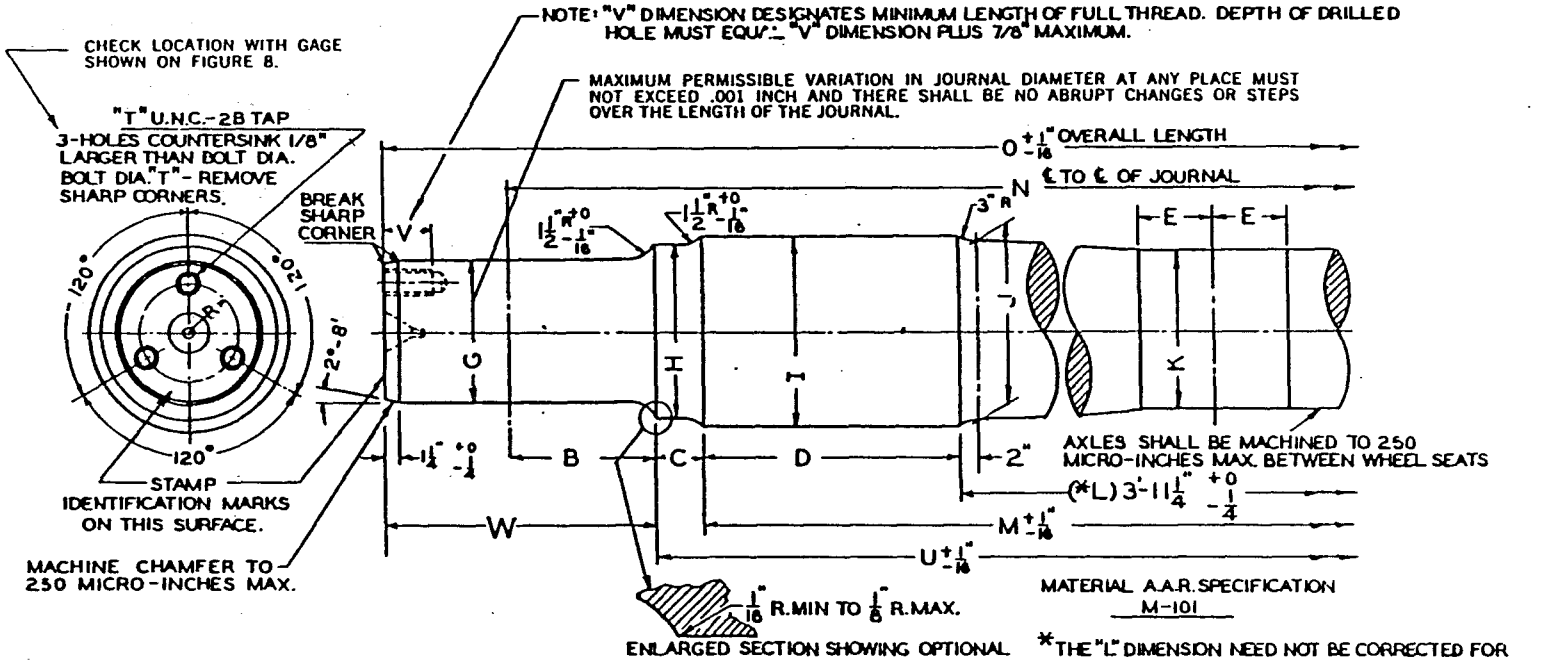
Table 1

Inside wheel boss Outside wheel boss Tight fitting band	Y N N	N Y N	Y N Y	N Y Y
$d = .0762 \sqrt[3]{\frac{Wb}{2} - \frac{Hh_1}{m}(x - b) + \frac{Hh_1 x}{l} + Hh_2 + \left(\frac{W}{2} + \frac{Hh_1}{m}\right) h_2 \tan\theta}$	1			
$d_1 = .0762 \sqrt[3]{\left(\frac{W}{2} + \frac{Hh_1}{l}\right) x}$		1		
$d_2 = .0975 \sqrt[3]{\frac{C}{2} x}$		2		
Execute Table 2		3		
$d = \sqrt[3]{\frac{\frac{Wb}{2} - \frac{Hh_1}{m}(x - b) + \frac{Hh_1 x}{l} + Hh_2 + \left(\frac{W}{2} + \frac{Hh_1}{m}\right) h_2 \tan\theta}{0.0982\left(23{,}000 - 0.3\frac{E}{d_1}\left(1 - \frac{d_1^2}{d_o^2}\right)\right)}}$			1	
$d_1 = \sqrt[3]{\frac{\left(\frac{W}{2} + \frac{Hh_1}{l}\right) x}{0.0982\left(23{,}000 - 0.3\frac{E}{d_1}\left(1 - \frac{d_1^2}{d_o^2}\right)\right)}}$				1
$d_2 = \sqrt[3]{\frac{\frac{C}{2} x}{0.0982\left(11{,}000 - 0.3\frac{E}{d_1}\left(1 - \frac{d_1^2}{d_o^2}\right)\right)}}$				2
Execute Table 2				3

Table 2

$d_1 \geq d_2$	Y	N
$d = d_1$	X	
$d = d_2$		X

Figure 21. ANZR Wagon Axle Design



CLASSIFICATION OF AXLE	SIZE OF JOURNAL	DIMENSIONS																
		B	C	D	E	G	H	I	J	K	M	N	O	R	T	U	V	W
B	4¼" x 8"	4⅜"	1⅜"	7⅜"	1½"	MAX. 4.004 MIN. 4.003	5"	6 3/16"	5 7/16"	4¾"	62½"	75"	83"	1 7/32"	¾" — 10	65¾"	1⅜"	8⅜"
C	5" x 9"	5 5/16"	1 7/16"	7⅜"	1½"	MAX. 4.6915 MIN. 4.6905	5⅞"	7"	6 3/16"	5⅜"	62½"	76"	84⅞"	1½"	⅞" — 9	65⅜"	1⅞"	9¼"
D	5½" x 10"	5½"	1¾"	7⅜"	3"	MAX. 5.1915 MIN. 5.1905	6⅜"	7 9/16"	6¾"	5⅞"	62½"	77"	86⅛"	1¾"	⅞" — 9	66"	1⅞"	10 1/16"
E	6" x 11"	5 15/16"	1 13/16"	7⅜"	3"	MAX. 5.6915 MIN. 5.6905	7.032 7.030	8¼"	7 5/16"	6 7/16"	62½"	78"	88"	1 15/16"	1" — 8	66⅛"	2"	10 15/16"
F	6½" x 12"	6 7/16"	1 13/16"	7⅜"	3"	MAX. 6.1915 MIN. 6.1905	7.532 7.530	8¾"	7⅛"	7⅜"	62½"	79"	89⅜"	2⅛"	1⅛" — 7	66⅛"	2⅛"	11¼"
G	7" x 12"	5 15/16"	2 5/16"	7⅜"	3"	MAX. 7.004 MIN. 7.003	8.002 8.000	9½"	8 9/16"	8"	62½"	79"	89¼"	2 5/16"	1¼" — 7	67⅛"	2¼"	11 1/16"

NOTE 1: "I" DIMENSION TOLERANCE IS +0.015 −0.015

NOTE 2: MAXIMUM ALLOWABLE RUNOUT BETWEEN "G" AND "H" IS .006" TOTAL DIAL INDICATOR.

NOTE 3: MAXIMUM ALLOWABLE RUNOUT BETWEEN "G" AND "I" IS .008" TOTAL DIAL INDICATOR.

NOTE 4: FOR 7" X 12" JOURNALS, END CHAMFER IS 4° 15' 5/8" ± 1/8".

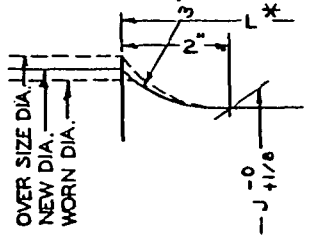
NOTE 5: REQUIRED FOR PRESS FIT OF ROLLER BEARING BACKING RING WHERE THE "H" DIAMETER IS TOLERANCED. FINISH TO 125 MICROINCHES MAXIMUM WHEN TOLERANCED.

NOTE 6: CHECK NEW AXLES WITH GAGE, WHEEL AND AXLE MANUAL FIGURE 5.26.

NOTE 7: RUNOUT ON JOURNAL SURFACE "G", WHEN ROTATED ON CENTERS, MUST NOT EXCEED .015" TOTAL DIAL INDICATOR.

NOTE 8: DIMENSIONS "B" AND "N" ARE CONSIDERED ENGINEERING DATA FOR TRUCK DESIGN AND DO NOT NECESSARILY AGREE WITH AXLE MANUFACTURING TOLERANCES.

ON NEW AXLES MACHINE 3 INCH RADIUS AT SPECIFIED DIMENSION "L" FOR ANY WHEEL SEAT DIA.



ENLARGED VIEW SHOWING CONDITION AT FILLET JOINING BODY PORTION OF AXLE TO INSIDE EDGE OF WHEEL SEAT.

Figure 22. Wagon Standard Axle Designs

gravity, load capacity, and the use of axle mounted disc brakes have resulted in more detailed investigation into the axle's performance. Sometimes axle performance is checked when the Traffic Branch wishes to use an existing wagon to carry an unusual load.

Axle stressing is based on the ANZR Manual of Standard and Recommended Practice (see fig. 21) which is derived from results of experimental work accumulated over many years. Insight into the formulas and design data is provided by Byrne (1968) and Spencer (1945). The axle stressing procedure is used in the Design Office to check that stress levels are acceptable at selected locations, mainly around the wheelseat (NZR CMEO Design Office : Nx, Ukx, Bf Design Notes), and is used to calculate the minimum diameter (NZR CMEO Design Office : Nk Design Notes) at positions along the centre line. To facilitate these calculations the Design Office has written a program, which includes disc brake load, for an HP9825.

However it is difficult to predict the actual stresses in an axle and their effect on the axle's life, the correct functioning of which is essential for safe and reliable transport, and so inspection (for example, ultrasonic testing) is performed.

The material specifications have been based on international standards.

The proportioning of geometry in the "standard" designs (see fig. 22) has evolved to counter a number of failure modes. Raised wheelseats, fillet radii, placing of identification marks and surface finish have been important in reducing crack propagation and fretting corrosion at the bearing and wheelseats. Surface heat treatments and cold rolling are also used in this regard. The taper in the axle body

gives more elasticity in the centre of the member, thereby reducing fretting at the wheelseat, the effect of rail/wheel impacts, and the axle weight. The wheels are assembled with a press fit which may be loosened by over-heating when braking and by "bell mouthing" (cone shaped wheel hub).

There is a range of bearing seat geometries and axle ends which are compatible with the standard bearing designs and end caps.

4. BEARINGS

(1) Function

Bearings give the wheelset their rotational freedom and transmit guidance and braking forces and vehicle weight between the suspension and axle.

(2) Bearing Types

All new wagons are assembled with roller bearings rather than plain bearings. This has been the case for many years although considerable numbers of wagons with plain bearings are still in service. Horger (1951) presented the case for the change from plain to roller bearings. Use was made of inservice performance data on both types and the evaluation involved consideration of relative costs of replacement (material and labour) of bearings, axles and lubricant; cost of inspection facilities and inspection labour; maintenance and storage facilities; bearings and other component weights; frequency and consequences of failure (fires, damaged axles and equipment damaged in derailments); labour and material costs in servicing wheels and removing

wheelsets from suspensions; cost of delays and time out of productive service caused by bearing failure; running friction and its affect on hauling capacity; starting friction and coupling slack required for starting (and as a consequence the cost of impacts due to slack); and so on.

The type 14 and type 16 three piece bogies use "package bearings" (a pair of roller bearings within a housing supplied as a unit by the manufacturer) whereas older freight bogie designs have a larger cast axlebox which contains the bearings. Some of the more recent four-wheel wagon designs (Nx, Nk) also have "package bearings".

(3) Design and Selection

Selection of bearings, then, is a matter of selecting a manufacturer's product with consideration of the usual factors of cost, standardisation within the organisation, and so on. The manufacturers supply a range of bearings that meet the various railway standard specifications, and calculation of bearing life expectancy must be based on the individual manufacturer's information.

Apart from fatigue failure (which results in surface cracks, flaking and shellout), brinelling due to coupling shocks is an important failure mode. Brinelling is the permanent deformation of the bearing races caused by contact stresses exceeding the elastic limit. Bearing cap screws, which hold the bearing components in place, can loosen and go missing. Inservice failures are inspected for and, during maintenance, wear is checked against limits set by the manufacturer.

(4) Bearing Housings

Axleboxes, bearing housings and adaptors are the devices used as

interfaces between the bearing and the suspension. In the case of adaptors they locate the bearing relative to a structural part of the suspension - the sideframe in three piece bogies and the spring saddle in Nx pedestal suspension. In other designs, notably those with primary springing, the axlebox or housing is located more directly by the spring assembly and sliding guides or rubber bushed rods. Examples of this concept are the four wheel wagon link suspensions and the traditional horn suspension.

Axleboxes have been developed over many years, are of cast steel and require seals and removable covers for access. The vertical faces of the axlebox (that slide against the vertical members attached to the wagon underframe) usually have replaceable liners attached to absorb wear. These liners are positioned in accordance with suspension travel.

5. REFERENCES

ARMSTRONG, J. (1981) Wheelsets : a world view. Railway Age, 182 (22): 41-45.

BRUNER, J.P. and others. (1967) Analysis of Residual, Thermal and Loading Stresses in a B33 Wheel and Their Relationship to Fatigue Damage. Journal of Engineering for Industry, Trans. ASME, 89 : 249- 258.

BUTCHER, C.F.G. and HORN, G.V. (1978) Wheel-rail Dynamics. Proc. I.E. Aust. Conference on Heavy Haul Railways, Perth.

BYRNE, R. (1968) Railroad Axle Design Factors. Journal of Engineering for Industry, Trans. ASME, 90 :33-42.

DEARDEN, J. (1960) The Wear of Steel Rails and Tyres in Railway Service. Wear, 3 :43-59.

FREDERICK, C.O. (1978) The Effect of Wheel and Rail Irregularities on the Track. Proc. I.E. Aust. Conference on Heavy Haul Railways, Perth.

HEWITT, G.G. and MUSIOL, C. (1979) The Search for Improved Wheel Materials. Institute of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.101-110. (I. Mech. E. Conf. publication 1979-11).

HOPPER, A.T., and others. (1977) Elastic Finite-element Stress Analysis of Railcar Wheels. Chicago, Illinois, AAR Technical Division. 106p. (AAR Research and Test Dept. Report No. R-268).

HORGER, O. (1951) If All Freight Cars Had Roller Bearings. Railway Age, Part I, Dec 31,1951, p. 37-42, and Part II, Jan 7, 1952, p. 63-66.

JOHNSON, K.L. (1982) One Hundred Years of Hertz Contact. Proc. Institution of Mechanical Engineers, 196 : 363-378.

JOHNSON, M.R. and others. (1977) Analysis of Thermal Stresses and Residual Stress Changes in Railroad Wheels Caused By Severe Drag Braking. Journal of Engineering for Industry, Trans. ASME, 99 : 18-23.

KALKER, J.J. (1979) Survey of Wheel-rail Rolling Contact Theory. Vehicle System Dynamics, 5 : 317-358.

KUMAR, S. (1980) Some Wheel-rail Interaction Factors Influencing Vehicle Dynamics. Rail International, 11(10) :599-610.

LONG, G. (1978) Wrought Wheel Design for Heavy Axle Load Unit Train Operation. Proc. I.E. Aust. Conference Heavy Haul Railways, Perth.

NEWCOMB, T.P. (1979) Thermal Aspects of Railway Braking.

Institute of Mechanical Engineers International Conference on Railway Braking, University of York, 1979. p.7-18.(I. Mech. E. Conf. Publication 1979-11).

NZR. CMEO Design Office. Bf Wagon Design Notes. (Anderson, P.J.) 1966.

NZR. CMEO Design Office. Nk Wagon Design Notes.[1980].

NZR. CMEO Design Office. Nx Wagon Design Notes. (May, D.J.) [1976].

NZR. CMEO Design Office. Ukx Wagon Design Notes.

PARK, Y.J. and STONE, D.H. (1981) Cyclic Behaviour of Class U Wheel Steel. Journal of Engineering for Industry, Trans. ASME, 103 : 113-118.

PAUL, B. and HASHEMI, J. (1981) Contact Pressures on Closely Conforming Elastic Bodies. Journal of Applied Mechanics, Trans. ASME, 48 : 543-548.

RADFORD, R.W. (1977) Wheel/Rail Vertical Forces in High Speed Railway Operation. Journal of Engineering for Industry, Trans. ASME, 99 : 849-858.

RUSIN, T.M. and others. (1979) Application of the Finite Element Method in the Development of Improved Railroad Car Wheel Designs. Journal of Engineering for Industry, Trans. ASME, 101 :378-384.

RYAN, C.F. and HUNDY, B.B. (1960) Steel Wheels and Tyres. Journal of the Institute of Locomotive Engineers, 50 : 304-344.

SPENCER, D.W. (1945) Notes on Axle Design and Performance. Journal of the Institution of Locomotive Engineers, 35 : 263-290.

THOMAS, T.J. and others. (1982) Fatigue Analysis of Railway

Freight Car Wheels. Int. Journal of Vehicle Design, 3(1) :90-102.

VINK, P. (1981) Derailment Statistics for Years 1979 and 1980. NZR Way and Works Branch Research Report 192.

WANDRISCO, J.M. and DEWEZ, F.J. (1960) Study of the Defects That Originate and Develop in the Treads of Railroad Wheels During Service. ASME Paper No.60-RR-1.

WETENKAMP, H.R. and KIPP, R.M. (1978) Thermal Damage and Rail Load Stresses in a 33 Inch Railroad Car Wheel. Journal of Engineering for Industry, Trans. ASME, 100 : 363-369.

WETENKAMP, H.R. and others. (1950) The Effect of Brake Shoe Action on Thermal Cracking and on Failure of Wrought Steel Railway Car Wheels. University of Illinois Engineering Experiment Station Bulletin No. 387.

WETENKAMP, H.R. and others. (1980) The Influence of Brake Shoes on the Temperatures of Wheels in Railroad Service. Journal of Engineering for Industry, Trans. ASME, 102 : 32-36.

CHAPTER VIII

WAGON DESIGN - GENERAL OBSERVATIONS AND DISCUSSION

In the preceding chapters an attempt has been made to describe the wagon design system, but perhaps more accurately it only describes a few processes identified in the records of some of NZR's wagon designers and a few of the relevant published analytical and experimental works.

However, these chapters do contain sufficient information to enable some general observations to be made about the characteristics of the wagon design process, characteristics which will be important in the design of computer-aids.

Design in general is an activity in which it is difficult to recognise order. There is no unique, unvarying methodology of design : the process is as much as anything an art. There are many possible paths to the end - the production of the artefact - the path chosen depending very much on the situation in hand and on the style of the individual designer. Unlike well defined systems all the activities to complete a wagon design can not be determined before hand - the examination of the design process in an historical sense will only provide an incomplete set. There is no right or wrong solution, rather many good (some better than others?) and many bad. No one can predict what the designer will see as the final design form and shape for the next design (although those in authority may restrict the choices). Usually there are no rules or guidelines to determine when the problem is solved nor can the product of the design process be definitively tested. The problem is important enough to warrant considerable

investment in time and money, is unique in some way (or else the solution that was used last time will apply this time also) and yet the problem cannot be definitively stated.

However some common characteristics (as suggested in the literature on design) in the design works have been identified.

1. AN ADAPTIVE SYSTEM

To begin with, wagon design is a set of activities performed by humans within an organisation; both the designer and organisation are purposeful systems. That is they can show adaptive and objective seeking behaviour (although the design management does not give the designer complete freedom of action in the design work).

Over longer periods of time any non-purposeful system is doomed to failure in the turbulent environment of railway wagon design. Adaptive behaviour of the wagon design system is required to maintain or improve performance when there are events in its environment such as rises in oil prices, deregulation of competing modes of transport, review of internal management policies, new goods to transport (for example, containers, bulk wine), new materials used in wagon construction, new suspension designs, new structural concepts (for example, stressed skin hopper wagons) developed outside NZR, new or revised codes of practice and specifications, and new understanding of wagon behaviour. Changes in relative importance of the objectives set by management force the wagon design system to design according to the operational requirements of the day. The wagon design system also monitors changes in itself, adapting to maintain or improve performance. The wagon design system

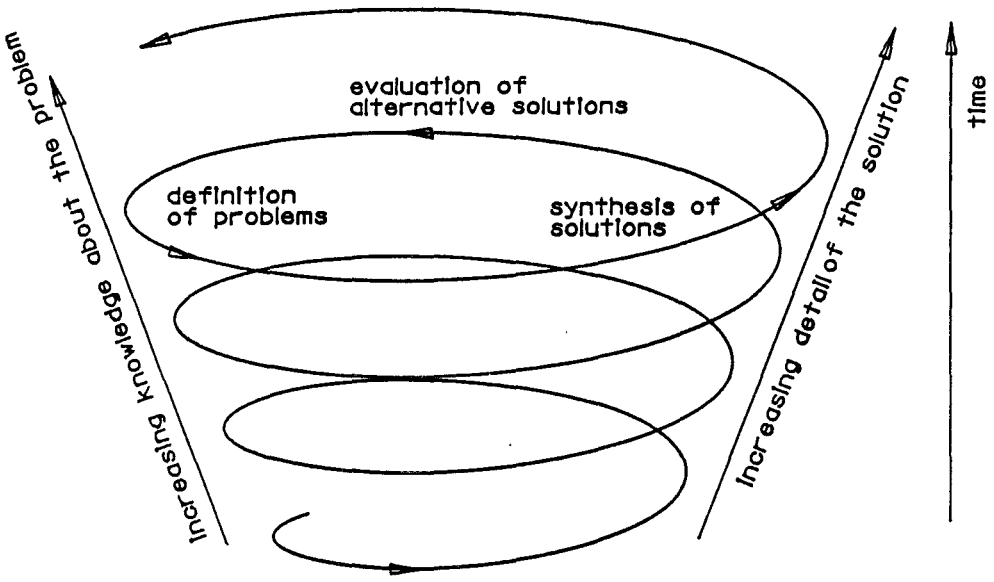


Figure 23. The Design Spiral

produces changes in itself or in its environment to effect these improvements.

2. AN INCREMENTAL ACTIVITY

Another commonly held belief about design is that it is an incremental activity concerned with how things ought to be in order to attain goals : as we refine our understanding of the problem and build up the implementation to address this new knowledge, we discover new requirements. This is in contrast to earlier suggestions (often prescriptive) that the design process is linear in nature, proceeding from specification to final product in a sequence of processes. Perhaps a better representation of design is one of a spiral (see fig. 23) depicting the growth of knowledge as one moves among the various activities, but even this is too ordered.

A central problem is that of obtaining the set of criteria that can be used as the basis of a designer's decision-making. Most criteria are only specified as predicates to be fulfilled and not as criterion readily quantified or easily defined. Criteria can be largely implicit as well as ambiguous. As a result of using incomplete and inexact information to construct an artefact, most artefacts must be continually modified to meet changing requirements and to meet refined requirements : no amount of initial design work is a complete substitute for actual use. Specification and implementation are inextricably linked in realistic situations and only artificially separated.

For many problems there is no other way than to plunge into activities which are not fully understood. If we are lacking detailed

specifications of the final goals, several alternatives have to be developed and compared with each other. Specification through modelling is a promising methodology for this class of problem.

Although not evident in the survey on wagon design as presented in this thesis, design is an iterative trial and error process composed of: organising and analysing information (manipulating information to produce new information); deciding and judging acceptability of solutions; deciding direction of design process (what is the next problem to tackle); and searching for ideas, suggestions, and alternatives. The synthesis process is repeatedly (and for the most part unpredictably) interrupted to analyse what has been done so far and to find the best way to proceed. These activities appear to be inextricably intertwined. The partially completed design serves as a major stimulus for suggesting to the designer what he should do next. New information is extracted from memory and reference sources, it is analysed and evaluated and so another step is taken in the development of the design.

The key parts of this evolutionary path are important to the understanding of the design artefact which is a prerequisite for effective modification.

3. MODELLING AND DESIGN DESCRIPTION

The designer by definition thinks about the artefact before it is built. He experiments with low cost manipulatable models which are abstractions of the problem - models that have detail stripped away to expose the essential character of the problem.

Models allow the production of a number of potential designs that are cheap enough to be thrown away and detailed enough to be tested and to be argued about.

(1) Models in the Design Process

A model is a representation of reality and there are many types, ranging from physical models to quantitative numerical models to conceptual models. In the early stages of design, information comes from previous personal experience, from manuals, reference books, existing working drawings, and observation of the wagon's physical environment. The early design lacks precision and detail, its practicability is yet to be proved. Sketches provide evidence for the existence of conceptual models at this stage. As the design progresses this information must be supplemented by specific information about the particular problem and proposals. The designer sketches and redraws in the manipulations of an explicit two dimensional model, a process which culminates in the production of final working drawings - part of the message describing the essential features of the new wagon/assembly/component. Drawings model form and appearance whereas mathematical models are concerned with other quantitative aspects. Gradually a description of the artefact is developed and its parameters adjusted so that the models give the desired responses to events in their environment. When adjustment is complete, the description is translated into reality and the artefact is built.

(2) Formalised and Implicit Models

Models may be explicit (conscious thought or formalised procedures) or implicit (subjective and intuitive). Designers, through education, and communication with other designers learn explicit models and explicitly formulated inferential procedures. They can also be developed by an individual designer if he reflects on his past behaviour and experiences. Implicit models are developed by experience. Models that are externalised and communicated in some sort of public language allow the relative truth of these models to be evaluated.

(3) Model Complexity

The complexity of a model depends on : the randomness and complexity of its environment; the relative value of the outcomes of the situation; the consequences of failure; and the degree of accuracy required in achieving the desired outcome. Whether the designer uses or develops a model partly depends on the expenditure of time and resources required and the time, resources and information that are available. Continual development of formalised models is a characteristic of wagon design.

The formalised models can be viewed in a number of ways. They can be classed according to their sophistication and level of detail - the variety of properties and interactions between properties that they include. For example, a model could be a single linear algebraic equation (a formula) or at the other end of the scale it may be a large set of nonlinear equations. For any formalised model (particularly, for mathematical models), there may be a number of methods for obtaining a solution. Increasing randomness may be dealt with by using probability

distributions instead of deterministic models. Clearly the level of detail in the model must be appropriate to that required. Elementary models are usually used initially and more detailed calculations are carried out when the design has been developed far enough for suitable data to be available. The choice of model and method of solution has implications for the time taken to arrive at a solution and the accuracy (with respect to nature) of the solution. The physical artefacts are much more complex than humans can deal with on their own and so the designer uses judgement and other means to limit investigations. More extensive and sophisticated investigations are possible with the aid of computers and other tools.

(4) Model Scope

The scope of a model is another basis for classification:

(a) Behaviour prediction is the analysis of a time varying system in which the variables describing the state of the model take on values as the simulation proceeds. By fixing all the design variables one can examine the consequent states in great detail. However all evaluation and generation or modification of the postulated design is external to the model. Changes in the state variables may be discrete and happen at irregular intervals, and the state variables or a subset of them are measures of the performance of the postulated design. In this sense behaviour prediction goes from physical properties to functional properties. Behaviour prediction includes such general models as Newton's Laws of Motion.

(b) Feasibility seeking involves the sorting of design postulates into satisfactory or unsatisfactory groups, but no other ranking is

applied: all decisions which conform to the rules are equally acceptable. The boundary of satisfaction is usually termed a constraint and may be regarded as a rapid change in the level of satisfaction with a small increment in the design variable - a step function (for example, stresses in a member are satisfactory till they become greater than some prespecified stress limit). The constraints may be prescribed by models of physical behaviour (for example, Newton's Laws) or may be prescribed more directly by man (for example, social laws, codes or recommended practice laid down by international or local organisations) to deal with experienced events but usually involving a fuzzily defined notion (for example, safety). These later constraints may be broken at some risk.

The feasibility seeking models of the design process may be able to generate a range of solutions as a consequence of the recursive application of a set of decision rules. If there exists a procedure for consciously inferring the properties of the design then it may be used to create, for example, the sizing or selection of set form as in parameterised geometry and standardised parts.

A design that is satisfactory with regard to some criteria is not necessarily an acceptable design as other criteria that have not been modelled may apply.

(c) Goal seeking models of the design process search the whole field of feasible solutions to identify those best suited to the stated goals. Decisions are made to modify the design according to their ranking on an explicit measure of effectiveness. Goal seeking models proceed from the initial state and a statement(s) of the goal (a functional specification, for example, minimum weight) to the specification of physical properties. The search techniques developed

in artificial intelligence and the mathematical optimisation techniques are examples of the methods used in goal seeking models.

In multi-objective function optimisation there are techniques to either graphically present the better solutions or to make decisions on the basis of trade off information between the objective functions which is supplied in advance to the model (Gero and Radford, 1980).

Also included in this type of model are the documented design hints, good design practices and so on. It is possible to enter these recorded beliefs of experienced designers into a computer so that it can automatically select the best known physical properties.

The use of optimising procedures has been restricted because in most design problems there are many vague and conflicting goals, and the definition of best depends on the designer's judgement. In the absence of complete and detailed functional specifications it is more realistic to define criteria for an acceptable or satisfying solution or in the absence of a quantified criteria to evaluate by feelings.

(5) Interrelated Models

Another aspect of complexity in modelling the design artefact is that in order to understand the artefact it is modelled as a number (in wagon design a large number) of individual models; the assemblage of the models presents a major problem. The data for one model can also be part of the data for another model, the output of which is the input to yet another model. There are difficulties in getting the assemblage of smaller models to adequately predict the performance of the real artefact.

(6) Description of the Design Artefact

Through the use of numerous models and evaluation of many alternatives a vast amount of information describing the design is amassed as the design proceeds. And when the information on completed designs is included, maintenance of consistency and ease of access become important issues.

Models in general can be used with numerical constants, boolean constants, alphabetic constants, and so forth or they may be manipulated symbolically (that is, with variable names) using the logic or the particular language of the field, for example, linear algebra.

A diagrammatic representation is often associated with a model. For example, draughting involves the use of working drawings which bear close resemblance to the artefact. At a more abstract level are symbolic representations of conceptual elements such as point forces, simple supports, built-in ends, springs and masses, pinned joints, resistors, and so on.

Graphics and summaries of attributes of selected entities are often used to reduce complexity in the design description for the human designer.

(7) Design Models Compared with Some Other Computer-based Models

Thus the designer's models differ significantly from airline reservation models, payroll models, etc. The designer uses implicit models and deals with a wide variety of choice situations (there is always by definition variation from one design problem to the next).

4. RESTRICTING THE DESIGNERS FREEDOM TO ACT

(1) Models

With research we can formalise implicit models. But if we restrict the designer to using only these models, then without freedom of choice he is not performing adaptive design. That is not to say that these models should not be developed rather that the designer, supervised by design management, should have control over the design process.

There are good reasons for allowing the designer to have control over his design - the models and procedures the researchers identify are often incomplete and will contain generalisations. There will be exceptions to these models, exceptions which the designer should recognise and be able to act on. The implicit models and assumptions change with time as the design system adapts to new pressures, new technology and so on.

The designer proceeds by making decisions in an environment in which there are often gaps in understanding and available information, in which requirements are poorly defined and conflicting, in which the constraints of time and resources are real and pressing, and in which the value systems are ephemeral and poorly understood (Elms, 1982).

The process by which designers review the quantitative information they have assembled and then make the qualitative judgement is, as yet, ill-defined, and much freedom must be left to the designer in doing it.

The interactions between the components of the artefact and between the artefact and its environment are complex and uncertain and the consequences of failure of the artefact are often catastrophic.

Design is a holistic process putting together complex situations unable to be described by any system of logic. The human mind has the ability to unconsciously integrate and weigh such complex situations so that it can forge progress in the design. It may, as well, add or omit criteria, or apply unfounded factors out of ignorance, personal prejudice or lack of attention. Designers also base their creative and evaluative actions on their entire experience.

Repeated use of a formalised model will influence the development of the designer's intuitive model, an important aspect of models that accurately predict complex interactions, but dangerous if they do not.

(2) Data Values

Because the designer deals with models and not the real thing there are often differences between design predictions and observed artefact behaviour. Again the experienced designer must be called in to judge whether the load conditions, material properties, etc. and the method of analysis are appropriate. By testing and observing the behaviour of artefacts previously modelled the designer can use models for the relative assessment of behaviour of one artefact compared with another. The designer can learn the effect of changing assumptions by exploring the solution space: he seeks information on how much a solution's performance will alter given changes in design parameters (sensitivity analysis) and on how stable is the choice of best solution given changes in the parameters (stability analysis).

(3) Benefits of Control

Control over designer's choices is a method of reducing the likelihood of the occurrence of human error, or directing the designer to improved achievement of management objectives. Examples are standard design procedures and standard parts.

Design management may control the design because:

(a) The designer is not experienced (that is, his intuition based models are not sufficiently developed).

(b) The choices do not fit in with management policies and directives. Unnecessary variety in wagon and wagon component design can be detrimental to achievement of management objectives such as minimising cost and achieving safety.

(c) The designer is not acquainted with or does not understand the formalised models in the wagon design system.

And of course professional associations and society at large imposes constraints on the designer's freedom to act.

5. REFERENCES

ELMS, D.G. (1982) How Safe Should We Build? In Keey, R.B. and others. Engineering and Society. Christchurch, University of Canterbury. p.74-98.

GERO, J.S. and RADFORD, A.D. (1980) The Design in Computer Design. Proc. of 2nd Conference on Computing in Civil Engineering, Baltimore, 1980. New York, ASCE. p.876-890.

CHAPTER IX

INTRODUCTION TO COMPUTING TECHNOLOGIES

This chapter introduces computer database and user interface concepts and the notations used in subsequent chapters. Relevant hardware, system software and application software are also briefly described.

1. USER INTERFACE

(1) Interaction Techniques

Designers may communicate with computers through a wide range of peripheral devices. Some of these devices are suitable only for entering information, some only for receiving and others for both functions; they may suit on-line or off-line applications. They also vary tremendously in cost and sophistication.

(a) Cards and Paper Tape. Punched cards and paper tape, once the main medium for entering data, have become less popular although they provide a permanent record.

(b) The Alphanumeric Terminal. The on-line use of the combination of keyboard and cathode ray tube is the most frequent means of interactive dialogue between the user and computer. In its most basic (and cheapest) form it is the alphanumeric VDU (visual display unit). The "Qwerty" keyboard with some additional keys is used to type in information which will be transmitted to the processor. A command language enables a user to access any function with a single input

string but knowledge of the commands is necessary for efficiency. On-screen hierarchical text menus on the other hand require little training but menu traversal can be slow and frustrating. Some of the interaction techniques mentioned under graphical workstations, such as function keys, can also be applied to text oriented devices.

In a screen formatted operation, headings or fixed data may be written in by the computer for defining areas on the screen, similar to the headings on a form. The user cannot access these areas, only the unprotected areas known as variable data. A screen formatting program may supply help information and standard values as default data and data validity and protection checks may be performed. Thus the user's job is made easier as he simply has to fill out the form that appears on the screen. But this type of interface programming is device dependent and requires detailed knowledge of the display terminals. There are, however, for some suppliers screen formatting program generators e.g. MIMER/SH, which simplify and speed up the programming and provide some device independence.

Other development utilities are available to assist in the generation of menus, on-line context sensitive help text and message text files for applications.

Display management has recently advanced to allow dynamic subdivision of the screen into a number of windows (or views onto processes and data), that can be selectively displayed, hidden, moved, scaled up or down as required.

(c) Graphics Workstations and Displays. At the other end of the scale to alphanumeric VDU's are high resolution, graphics displays offering full colour and shading and providing rapid interaction between

operator and display. Intelligent terminals incorporating graphics memory and display processors enable the host computer to off-load functions thereby freeing the host for other tasks and improving system response as perceived by the user. Local mass storage and software may be sufficient for standalone operation.

Between these extremes are intermediate cost graphics terminals, the characteristics of each device suiting particular applications.

There are other media for visual display of information including the solid state displays (plasma panels, liquid crystals and light-emitting diodes) but they are not widely used. Some work has also been done on the generation of three dimensional images.

Means of interaction other than the traditional keyboard are available. Special function keys may be added to a keyboard for frequently used commands, keystrokes and so on. Predefined functions may be invoked by using a puck or stylus on a menu sheet on top of a electronic tablet or worktable. On-screen graphical icons and pop-up menus have become popular in recent years. Means of indicating spatial position on the display include: puck and tablet, touch-sensitive screen, cursor keys, light pen, joystick, electromechanical cursor. Spatial position may be used to input/output co-ordinates or to indicate a choice in menu display, or indicate selected pieces of the object displayed. Digitizers are often used for input of detailed geometry.

(d) Hard-Copy Output. Hard-copy techniques (printers, plotters and photographic methods) produce for the user the traditional permanent record of information which can be visually inspected.

Line printers are widely used for printing alphanumeric data and simple graphics. They are available in a wide range of prices, and

performance and functional specifications. Where noise reduction is important non-impact printers (ink-jet, laser, thermal printing, electrostatic, electrophotographic) are used. Some of these technologies are also suitable for graphics work. Their printing/plotting speed can be many times faster than impact printers or pen plotters.

A fast output can be achieved using COM (computer output microfilm). Thus it is possible to avoid large volumes of paper by using microfilm as the standard medium, and using pen plotters and printers for check output. Plotters come in a wide range of accuracy and resolution.

(2) Interactive Interpreters

(a) Compiler or Interpreter. A compiler is a computer program which translates another program called the source program into yet a third called the object program. The source programs are written in the source language, the object program produced is expressed in the object language. A "true compiler" translates all source language statements into the computer's own language (binary machine code) and then executes these translated instructions, whereas a "true interpreter" decodes and actions each instruction in sequence. The disadvantage of interpreting is that if an instruction is repeated several times, the decoding must be repeated equally many times. On the other hand compiling produces object code which is fast to execute, but translation of the source program is more time consuming than just decoding, and the compiler program size is bigger.

The balance between these two approaches depends on the relative time taken for decoding and action. If the action is relatively large,

the decoding overhead is relatively negligible. Because of the overheads of interpretation of a source language, almost all interactive compilers translate the source language into an internal language which is easier to decode. The closer the internal language is to the source language the closer the compiler to "true interpreter". At the other extreme when the internal language is close to machine code, the compiler is nearer a "true compiler".

Each source program solves a particular problem for a user. The source language should be one in which users find it easy and natural to solve their problems, and the object language, whilst probably quite opaque in meaning to users, will be a natural one for some machines to execute. Thus a compiler or interpreter can be viewed as a tool which transforms programs from the user's domain of problem solving into the machine's domain of problem executing without varying the semantics of the programs.

An internal language that is different from the "native" language of any particular computer enhances portability. This conflicts with efficient code: the more efficient the code becomes the more specialised the code generation has to be thereby losing its generality of application.

(b) Compilation Phases. Common to most "compilers" are the tasks of analysis of the source program and the synthesis of the object program. Analysis is usually performed in two distinct phases called lexical and syntactic analysis.

The lexical phase contains recognising the basic syntactic symbols in a string of input characters. These basic symbols include keywords (or reserved words) such as "write", operator symbols such as + and /,

assembly of literals such as "1.57" and "true", identifiers such as "y", delimiters and punctuation marks.

Syntactical analysis is performed on the basic symbols returned by the lexical analysis. Using syntactic rules which define the language, the compiler accepts input which conforms to these rules or indicates invalid input as containing syntax errors.

In summary the basic functions are:

- scanning the input for basic symbols;
- generation of a compact internal form of this input;
- a syntactical analysis of this internal form.

Additional processing of the language is semantically and possibly implementation dependent.

The third phase is that of code generation and is synthetic rather than analytic. It produces object code based on the relationship of the parts of the program to each other and to the whole program.

(c) Multiple and Single Pass Compilers. Multi-pass compilers have the disadvantage that the information transmitted from one pass to the next is encoded in some way and often written out to backing store to save space. This is slow and requires extra programming, although less computer main store is required as only one pass need be present at any one time. By combining the lexical and syntactical analysis and code generation phases into the same procedure or pass, the need for explicit generation of some information is eliminated. The requirement then for more main store is not generally a problem on modern day computers, and the one pass analysis and code generation enables faster compilation. In a virtual storage environment performance may be enhanced by splitting the compiler code into small sections that are

heavily used and then discarded, as each pass in a multi-pass compiler might be. However this is a strategy more for batch compilers than for interactive interpreters. This is because for each line typed, the interactive interpreter would need to go through every pass, whereas a batch compiler can process the entire source program as one unit.

(3) Grammars And Languages

(a) Metalanguages. A means to precisely and easily define the syntax of source language is required. A "metalanguage" is the language used to describe another, the most commonly used in definition of languages for computers is Backus Naur Form (BNF) notation.

The grammar (or syntax) of a language using BNF notation consists of a set of productions of the form:

"LHS" :: = "RHS"

The symbol "::=" means "is defined by". The left-hand side (LHS) and the right-hand side (RHS) are composed of two components:

(i) terminal symbols: a finite set of symbols which form the basic characters or groups of characters which can be strung together to form "texts" of the language, for example, the digits (0, 1, 2, 3, etc) and letters (A, B, C, etc).

(ii) non-terminal symbols: another finite set of parts of the language being defined by BNF phrases. The set of non-terminal symbols and terminal symbols must be disjoint for obvious reasons. Non-terminal symbols are defined in terms of other non-terminals and/or by terminals. For example, a number is a string of digits or a name is a concatenation of letters. To distinguish non-terminals, they are enclosed in angled brackets "<" and ">".

(b) Grammars. The rules which show which transformations of strings of terminals and non-terminals are legal are known as "productions". In the production or legal transformation "P ::= Q", the string P must contain at least one non-terminal. Hence a string of only terminals cannot be replaced by something else, and this is why they are known as terminals. If several productions have identical left sides then they may be combined by placing a vertical bar "|" between their right-hand sides. Thus "|" stands for "or", e.g.

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Chomsky (1959) has categorised grammars into four classes, each one essentially a subset of the previous one. Type 0 and Type 1, unrestricted productions and context sensitive productions respectively, do not have efficient compiling methods at present, even though they are needed to completely describe the syntax of most useful computer languages. Usually contextual constraints (for example, type rules) on a language are expressed in an informal manner, and the syntax is then described with a Type 2 or Type 3 grammar. Type 2, context free, restricts the LHS of the production to a single non-terminal, the RHS can be any string of terminals and/or non-terminals. Type 3, regular grammars, have the same restriction on the LHS as Type 2, but restrict the RHS to a terminal optionally followed by a single non-terminal. Regular grammars are useful in describing the lexical structure of a language.

Recursive definitions occur when a non-terminal is defined in terms of itself.

<variable name> ::= <letter> | <variable name> <letter>

Braces "{" and "}" are used to enclose repeating sets of symbols.

<variable name> ::= <letter> { <letter> | <digit>}

(4) Syntax Analysis With Particular Reference to LL(1) Grammars

(a) Parsing. The productions may be used to transform the start symbol (a non-terminal, usually the <program>) through a sequence of transformations to the basic symbols of the program. When a compiler attempts to parse (that is, describe the syntax) a program using this process, it is called generative or top down parsing. Alternatively the compiler may start with the terminals, perform a succession of transformations till the start symbol is reached. This process is known as recognitive or bottom up parsing. Bottom up methods will not be considered further; details of these methods are available in texts on compiling.

It is desirable to have an efficient parser that is easy to modify in order to incorporate extensions to the language. Table driven methods provide for ease of altering the grammar. From grammatical rules a table is built which is used to control a central parsing loop.

Another popular method for top down compiling is recursive descent. However for ease of implementation it requires that the encoding language support recursion, that is allows a procedure to call itself.

Most practical parsers scan the input from left to right and for all but the trivial cases it is less complicated to use the left-most derivation, as opposed to the right-most derivation. When the left-most derivation is used, on the basis of the current input basic symbol it is always the left-most non-terminal which is the current candidate for expansion using some production rule. For left-most derivation,

grammars with left recursive productions must be avoided because of the infinite loops that they introduce when attempting to parse programs. If selection of which production rule to expand with is not deterministic, "backtracking" may occur. This is inefficient as not only will basic symbols be "unread" and parsing "undone" but also generation of code and symbol table entries may have to be "undone" and in cases "done" again.

(b) Parsing of LL(1) Grammars. A top down parser which can make a deterministic decision about which alternative production to choose, given the next terminal and the history of the parse done so far is called an LL(1) parser. The first L stands for "Left to right scanning", the second for "using the Left-most derivation", and the grammar an LL(1) parser scans is an LL(1) grammar.

A formal definition of an LL(1) grammar is as follows (Tremblay and Sorenson, 1982a).

"A grammar G is LL(1) if and only if:

for all rules $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$

1, FIRST $(\alpha_i) \cap$ FIRST $(\alpha_j) = \emptyset$ for all $i \neq j$ and, furthermore, if $\alpha_i \stackrel{*}{\Rightarrow} \epsilon$, then

2, FIRST $(\alpha_j) \cap$ FOLLOW $(A) = \emptyset$ for all j.

... Given some string $\alpha \in V^*$, the set of terminal symbols given by FIRST (α) represent the left-most derivable symbols of α and this set is given by the equation

FIRST $(\alpha) = \{w | \alpha \stackrel{*}{\Rightarrow} w \ldots$ and $w \in V\}$

The FOLLOW sets are defined for a nullable non-terminal A (one which can produce the empty string)...

FOLLOW (A) $=\{w \in V \mid S \overset{!*}{\Rightarrow} \alpha A \gamma\}$ where $w \in$ FIRST (γ) and S is the start symbol of the grammar."

In the above formal definition of an LL(1) grammar:

$\emptyset$ is the empty set, ϵ represents an empty right-hand side of a production, $\rightarrow$ stands for the same as the BNF metasymbol ::=, $\overset{*}{\Rightarrow}$ is the "is derived from" relation or reflexive transitive closure of "=>", where "=>" symbolises a transformation or mutation using a language's grammar e.g. $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \alpha_n$ or $\alpha_0 \overset{*}{\Rightarrow} \alpha_n$. V is the set union of the set of terminal symbols and the set of non-terminal symbols, and V^* is the reflexive transitive closure of the set of symbols V under the operation of Cartesian product, that is, a string of symbols from V of any length.

The parser must decide which of the possible right-hand sides the left-most non-terminal should be expanded to, merely by looking at the next input symbol. The FIRST set of each right-hand side tells us which input symbols can be matched by that production, if the non-terminal is expanded to that RHS. The FIRST sets for a given non-terminal must be disjoint or else the parser will not know which production to use when a common element is the input symbol.

FOLLOW sets need only be defined for those non-terminals which can produce the empty set. The FOLLOW set is such that if there is a RHS AB, and A can be expanded to the empty string, then FOLLOW(A) gives the left-most derivable input symbols that can be produced from B. For the same reason as above the FOLLOW set must be disjoint from all the FIRST sets of a given non-terminal.

The FIRST and FOLLOW sets form the "director sets" for the productions. The other set in the partition is the error set; if any of its members appear as the input symbol the parser knows an error has

occurred. Further discussion on FIRST and FOLLOW relations is available in the texts on compiling, for example, Tremblay and Sorenson (1982b), and Davie and Morrison (1981) (who give details on elimination of left recursion, and transformation of non-LL(1) grammars into equivalent LL(1) grammars).

(c) The LL(1) Parsing Function. Recall that the parser expands the left-most non-terminal in order to match the input symbols. The string of input symbols is steadily eaten into by being read one at a time. As the parse proceeds, new terminals appear at the left of the transformed string and are matched with terminals read in. At this stage they can be discarded so that the parser is always trying to expand the left-most non-terminal in order to match it against the next section of input.

Tremblay and Sorenson (1982b) define the parsing function M(x,y), where x is the terminal or non-terminal currently under consideration, y is the input symbol to be matched.

"M is defined as follows:

1, $M(a,a) = \text{'pop'}$, for all $a \in V_T$

2, $M(\#,\#) = \text{'accept'}$

3, $M(A,a) = (\alpha,i)$ for all $a \in \text{FIRST}(\alpha)$ where $A \rightarrow \alpha$ is the ith production

4, $M(A,b) = (\alpha,i)$ for all $b \in \text{FOLLOW}(A)$ where $A \rightarrow \epsilon$ is the ith production

5, $M(z,a) = \text{'error'}$ for all $z \in V \cup \{\#\}$ and $a \in V_T \cup \{\#\}$ such that none of 1, 2, 3 or 4 apply"

Where V_T is the set of terminal symbols, # marks the end of the input string. The "accept" action indicates a successful parse, the

"error" action indicates an error is present in the input string. "Pop" occurs when the top element in the parse stack should be taken off and the next input symbol should become the current input symbol. The (α, i) means the current non-terminal should be expanded to the right hand side of production i, that is, string α.

2. INFORMATION MANIPULATION SYSTEMS

(1) Computer Hardware and Systems Software

The purpose of this section is to discuss in general terms the available hardware and systems software relevant to the design of the CAD system.

First a definition: an operating system is that set of software procedures provided in a given computer system to control the operation of the hardware and provide at least basic services in relation to the initiation and stopping of processes and transfer of data between the processor and peripherals. Additional facilities providing features such as file security may be provided in some operating systems.

A wide range of computers are available, their different abilities lend them to application to different areas. The following introduces some hardware and systems software technologies.

(a) Storage Hardware. Typical secondary storage mediums are magnetic tape and magnetic disc. Magnetic tapes are relatively cheap and therefore provide an economic way of storing copies of data and for archiving data, but are unsuitable for use with on-line systems due to the length of time taken to locate the required data items when they are not inherently sequential. Direct access devices such as magnetic discs

are usually used to overcome this limitation. Large systems can have disc storage capacity of many thousands of millions of alphanumeric characters (megabytes).

(b) Virtual Storage Systems. Although other techniques exist (e.g. memory overlaying) virtual storage allows extension of addressable memory to include secondary storage as well as primary memory, thereby allowing programs to reference addresses that need not correspond to real addresses available in primary memory. Using software or hardware or both, they allow users to create programs independent (for the most part) of the constraints of primary storage and facilitate the operation of multi-user systems. For example PRIMOS allows 32 megabytes of address space per user program, while in Data General AOS/VS each process (a collection of program tasks) can address 512 megabytes of the total four gigabytes of logical address space.

In today's virtual storage systems, binding (the association of instructions and data items with particular storage locations) is performed dynamically as a program executes. They also create load images that are preserved on secondary storage, enabling immediate loading without the often extensive overhead of recombining program pieces as in linking loaders. Virtual storage means that loading and unloading of program sections and data is automatically and efficiently handled.

(c) Multiprogramming and Multitasking. Process-oriented multiprogramming systems control multiple user and system processes independently and concurrently and protect processes for each other's activities. A process is an executing object of finite life defined to the operating system with an assigned set of privileges, access rights,

and resources and an execution priority. A process manipulates data obeying a program which may in turn transfer control to another program or spawn another process or task. Resource or access requests that do not receive privilege authorisation are actioned by the operating system.

Multitasking controls the concurrent execution of multiple tasks or subprocesses through a single process. The tasks can run in parallel or interact using task calls designed for execution control, communication and synchronization, and timing. An interactive user can swap to another task if the present task does not require continuous intervention.

Processes can communicate with multiple free format messages of variable length. Processes sending messages can continue activity without delay. The messages are spooled if no receiver is waiting for the message. A task that wishes to receive a message can be suspended until another process sends the requested message.

Shared program code is a single copy of a program that many users can use and is a mechanism for reducing memory requirements. Templates can be created so that a shared procedure can be linked to other application programs before being run.

(d) Time Sharing Systems. A time sharing computer is a system in which the computer system's processor is connected to a number of conversational terminals. With the drop in cost of hardware, selection of time sharing systems on the grounds of utilisation of hardware has become less important although where concurrent sharing of files amongst users or large main memory or high speed execution is required they can offer a cost effective solution. If the time sharing system becomes

overloaded they respond more slowly, which is annoying to users and is often the ground for breaking up and distributing the computing power closer to the user.

(e) Networked Systems. Networks are systems in which functions are distributed amongst the geographically dispersed computers, and resources (including data) are shared through communication links. For example, the preparation of input data using graphics can be performed on a workstation and when complete sent to a larger capacity mainframe for analysis. The results of the analysis can be sent back to the workstation for local processing. Low entry costs, cost-effectiveness, less dependence on the reliability of a single processor, increased specialisation, and less operating system overhead are some of the reasons quoted for the popularity of distributed processing. The appearance of high speed, 32 bit, multitasking, virtual storage single user workstations with integral networking capabilities has sparked much interest in the CAD popular press.

A distributed database is a database that is physically spread across a network of computers. Distributing the data aims to combine the efficiency of local processing (with no communication overhead) for most operations together with all the advantages of centralised control. In such a system each data item is stored at the location in which it is most frequently used, but remains accessible to other users in the network.

(f) Parallel Processing. Massive parallelism (systems with large numbers of processing units that function simultaneously) is currently being developed. Languages are being developed to exploit concurrency of execution, and hardware and operating systems are being designed to execute concurrent sections of programs more efficiently. These

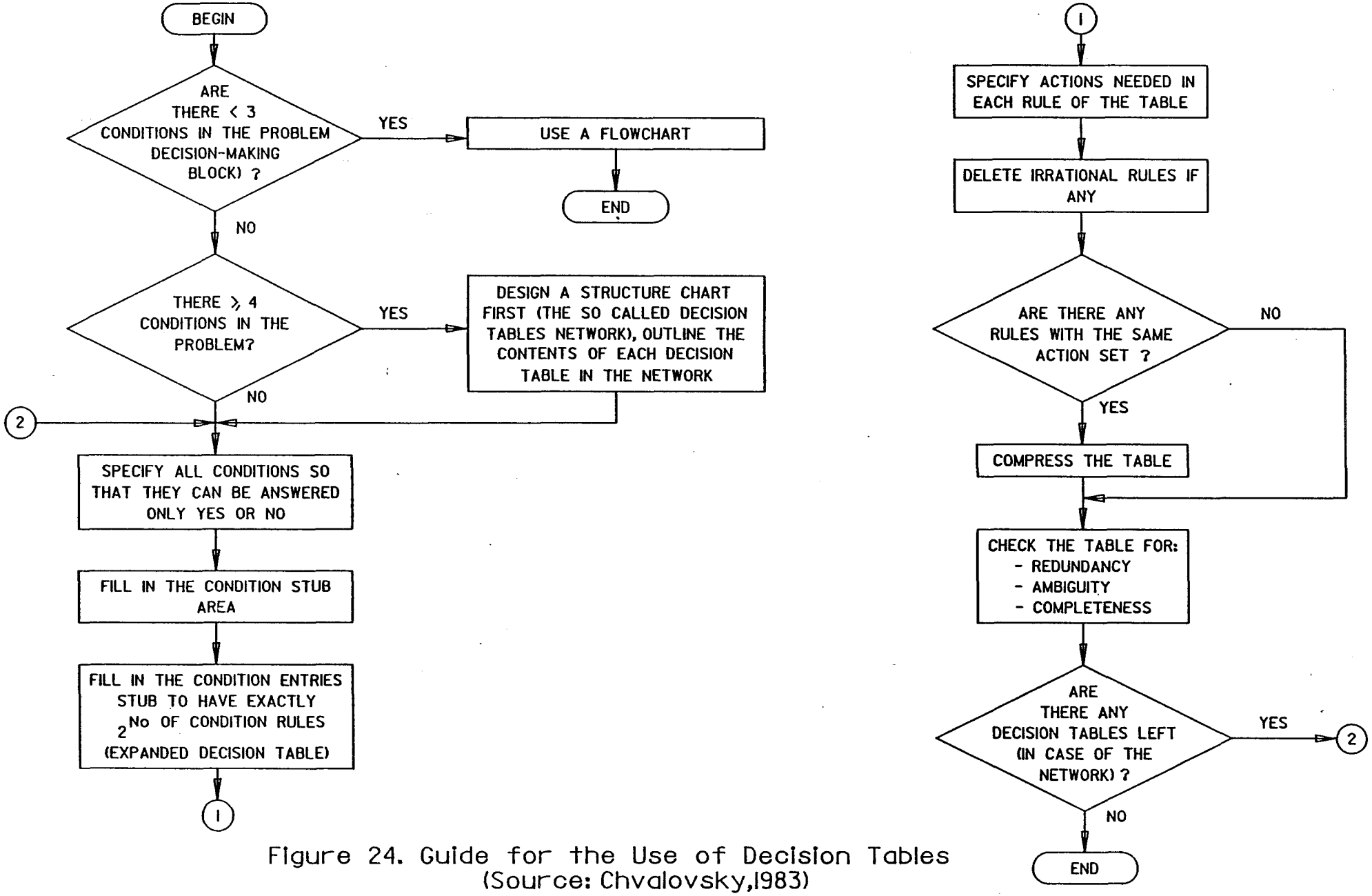


Figure 24. Guide for the Use of Decision Tables
(Source: Chvalovsky,1983)

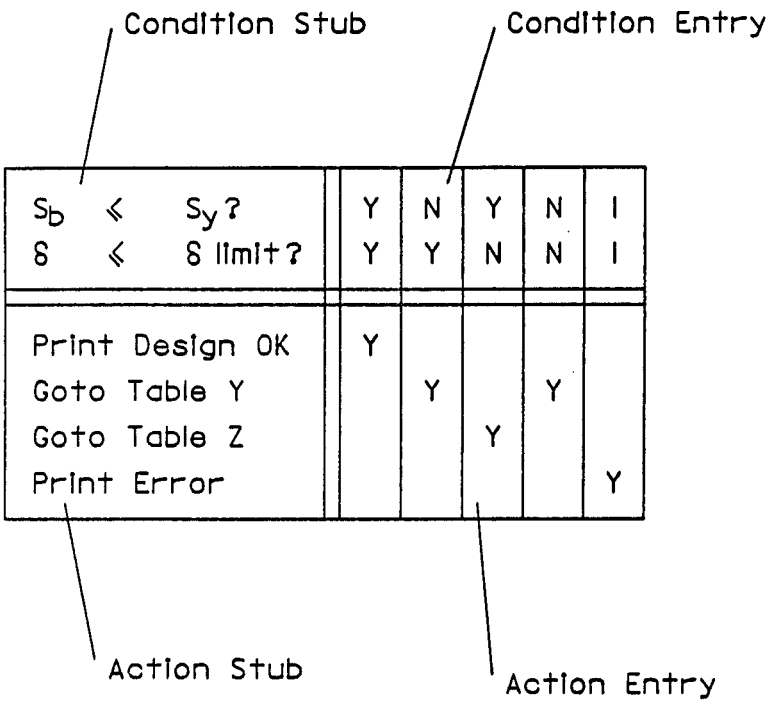


Figure 25. Annotated Decision Table Example

developments will have a significant effect on much more than just execution speed.

(2) Decision Tables

Decision tables provide a means for clear and consistent documentation of procedures particularly those with decisions based on multiple conditions (see fig. 24) and have been used as a high level programming language, (Chvalovsky, 1983; Fenves, 1976; Goel and Fenves, 1971; BS 5487; Hunt, 1983). As they have application in the formal representation of design requirements, this section briefly describes decision tables and their use. More detailed information is available in texts such as McDaniel (1968) and Pollock (1971).

Decision tables provide a concise tabular presentation of the logical relationships between the class of a state, expressed numerically or logically, and the action or actions to be performed when that stage occurs (see fig. 25).

The "condition stub", that is the upper left-hand section of a decision table, lists all the conditions to be considered in describing a problem. Each condition has a finite set of values or "states". For example, a condition may be "Sb $\leqslant$ Sy" and the states (or values) of the condition "Sb $\leqslant$ Sy" may be Y(yes), N(no) or I(immaterial). The "condition entry", that is the upper right-hand section, specifies the relevant combinations of the states of all the conditions.

A "rule" is a unique combination of condition states (one for each condition) and actions to be taken as a consequence, that is a vertical column in the entry section of a decision table.

The "action stub", that is the lower left-hand section of the

table, lists all the relevant actions (an action may, for example, print a message or assign a value to a variable or execute another decision table). If a rule is applicable, that is if the state of the complete condition expression (the condition stub) agrees with one of the available vertical columns in the condition entry, the entry or entries in the "action entry" indicate which action or actions symbolised in the action stub should be initiated. If the action or process is not to be executed, the corresponding box in the action entry is left blank. Decision tables which use Y, N, or I in the condition entry are usually referred to as limited-entry tables. An "else rule" specifies the action(s) to be taken for any combination of condition states not covered by the rules in a table; else rules are generally used to detect otherwise undefined error states.

(3) New Application Techniques

Although not yet evident in current wagon design practice, commercial equation solvers and expert systems are likely to be used in wagon design in the future.

General-purpose "spreadsheet" tools like Visicalc (from Software Arts) demands no programming other than the definition of equations. Each variable is displayed in a particular location on the screen and the effects of changing an independent variable are immediately visible. Other systems (MSC/CASE from MacNeal Schwendler Corp.; Math Pak from Cognition Incorp.) allow the use of predefined common engineering handbook equations, or vector and matrix variables (MSC/MATE). Recently these equation solving programs have been given increased flexibility so that there is now no discrimination between

dependent and independent variables (TK!Solver from Software Arts; Konopasek and Papaconstadopoulos, 1978). Equations may be entered into the system in the form most convenient to the user. The value allocated to any variable may be changed irrespective of its position in the equation and the effect on all related variables is then displayed.

The application of knowledge engineering (the acquisition, representation and manipulation of human knowledge in symbolic form) to design has led to the development of "expert systems". Expert systems have been defined as computer programs which use symbolic inference procedures to deal with problems that are difficult enough to require significant human expertise for their solution. An expert system is composed of:

- an inference engine which carries out the reasoning tasks;
- a knowledge base containing the domain specific knowledge;
- an explanation facility to explain how a conclusion was reached, why an answer is needed at a particular point or how a question can be answered;
- a state description containing the facts which have been inferred to be true and those found to be false in a particular session;
- a user interface.

Many expert system "shells" are available for expert system development. A shell provides the above components in a form independent of application domain.

Expert systems have been more successful in well bounded and defined areas where information can easily be described or quantified and in areas where there is agreement on what is correct (that is, an individual's style is irrelevant).

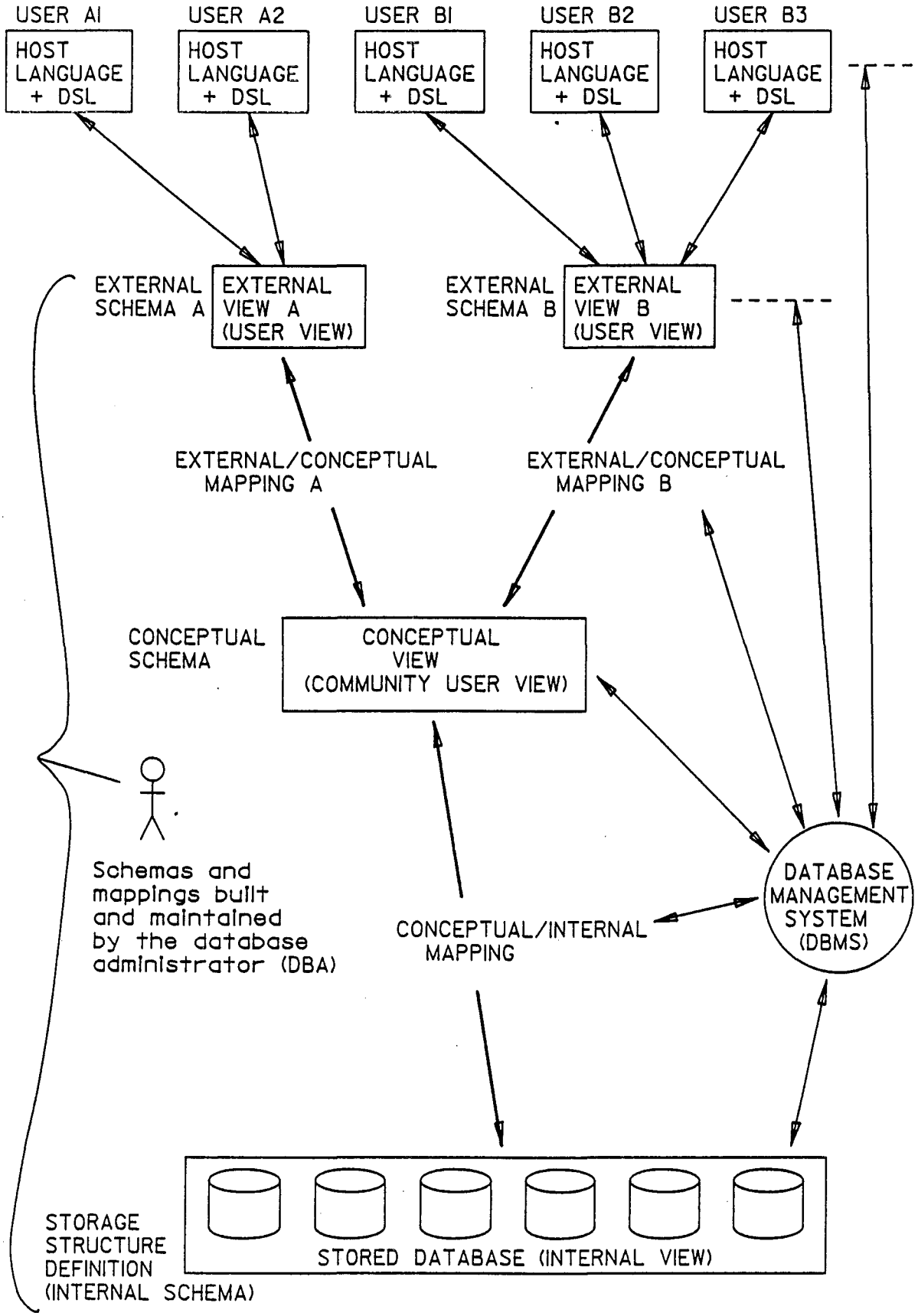


Figure 26. Data Views (Source: Date,1981)

3. DATABASE SYSTEMS

A Database Management System (DBMS) is a software implementation that maps logical data structures into physical data structures.

(1) Data Views

It is widely accepted that there are at least three levels at which data in a database system can be viewed (refer figure 26) (Date, 1981; Martin, 1977; Atre, 1980). (The author follows Date's (1981) terminology closely. Martin (1977) compares some alternative terms.)

The external level is that of the individual users' views, be they application programs or interactive users. (The application program may present data to an interactive user - yet another level, but it is outside the scope of the DBMS.) At the conceptual level is the user community view. Whereas the many external views each consist of some portion of the total database, the single conceptual view is of the entire database. There is also a single internal view, representing in machine oriented constructs, the database as physically stored.

(a) Languages. The languages employed by the system developers to access the database (for example, procedural languages COBOL, PL/I, FORTRAN) are referred to as host languages and the interactive non-procedural languages employed by the wagon designers (for example, query languages, report program generators, special purpose application based languages) are referred to as self contained languages.

A query processor is in effect a generalised routine which is particularised to a specific application (that is, the user's query) by the parameters (data names, Boolean predicates, etc.) appearing in the

query. Query facilities are more advanced than most early report program generators in that they provide on-line (as opposed to batch) access to databases (as opposed to individual files).

Modern report writers make data selection, sorting, grouping, formatting, statistical calculation and printing for predefined inquiries a relatively straight forward task.

Each language includes a data sublanguage (DSL) concerned with database objects and operations. Each data sublanguage generally is a combination of a data definition language (DDL) providing facilities for defining database objects and declaring attributes of structures (it enables DBMS to perform type checking and name resolution implicitly) and a data manipulation language (DML) providing features for specifying the operations to be performed on instances of database objects (see chapter XVI, sect. 1(7) for MIMER/QL examples). The DML may consist of, for example, a set of call statements to standard routines (provided as part of DBMS) or a set of extended application language statements.

(b) External Views. Each external view is defined by means of an external schema, which consists of definitions of the user oriented constructs. (Following Date, a "view" is a set of occurrences, in other words what data is observed, and "schema" is the definition of a view specified at a more abstract level).

As the database grows and changes, the conceptual schema must grow and change accordingly. Growth of the structures in the database may consist of addition of attributes (data items that characterise an entity) to entities (records containing information about a thing, place, event or concept) or the inclusion of a new type of entity. External views can insulate users from these types of changes.

Restructuring the conceptual schema (that is, the allocation of attributes to entities is altered) is more troublesome. Retrieval operations may remain unaffected using external views, but other operations, for example updates, may no longer work as before because of an inability to map updates onto conceptual schema.

With external views the same data can be viewed by different users in different ways. External views can also simplify the user's perception of the database and consequently the user's DML operations. In effect, complexity is moved out of the DML and into the DDL. Use of external schema also hides other data from the user thereby providing a simple automatic means of security control.

The schema is written using the DDL portion of the data sublanguage, and could take many forms. For example, it may be a declaration in the application programming language, or in an independent language, or using a facility provided by the DBMS.

(c) Conceptual View. The conceptual schema (the definition of the conceptual view using a DDL) describes the information entities of the enterprise, the relationships (or associations) between the entities and the information flow.

The conceptual schema is intended to be the description of the data "as it really is", that is, independent of any hardware, of the way in which it could be physically stored or accessed, of any particular DBMS or of any particular external view.

The external schema could differ from the conceptual schema in many ways. For example, the sequence of data items in a set of data items may be different; vectors may be redefined as multi-dimensional arrays; relationships between data items may be different; characteristics (e.g.

format) of data items may be changed; data items may be omitted.

The conceptual schema contains text describing a data element's meaning, use and units of measure. Every data element has a unique name, and synonyms and homonyms must be identified. All relations in which the data element participates must be identified including virtual relations and their method of derivation.

But the conceptual schema is more than just a collection of the application's data definitions as it includes many additional features such as the authorisation checks and validation procedures, audit controls; definition of users; data flows, how the data is created and used, by whom, by which programs, frequency of use, and for what purpose; redundancy and update propagation; language, function and mode of programs and transactions; and so on.

Few systems today - if any - support this level of comprehensiveness and independence. They may support some authorisation procedures, some physical data independence but little logical data independence and the conceptual view is little more than the union of all of the individual users' views. Minimally the conceptual schema includes data element groupings with key data elements and relationships between groups. Even if the DBMS system available is such that the conceptual schema will exist only in manuscript form then the enterprise will still benefit from having a self-contained succinct description of its operational data. If it is expressed in precise human oriented (rather than machine oriented) terms then communication among the many functions involved (end-users, management, application programmers, database administrator and so forth) is so much easier.

Atre (1980) regards the logical schema as the subset of the

conceptual schema that is mapped to the DBMS to be used and the external views supported.

(d) Internal View. The internal view is one removed from the physical level, since it does not deal with physical records, blocks, or any device-specific constraints. However the internal schema does define the stored records (e.g. precision, length ,left or right justification, character or floating point, and so on of the record fields), what indexes exist, what physical sequence the records are in, where the data is, and so on. The mapping of the internal schema to the physical storage is implementation specific. The various techniques used are not discussed any further here as they are covered in detail elsewhere (see for example, Date, 1981).

Thus if the internal schema is changed in any way, the logical to internal mapping must be changed accordingly, so that the logical schema may remain invariant. Keeping the views apart and employing a mapping function will ensure that changes to any schema will not necessarily affect any other schema, the changes being absorbed within the mapping functions. Figure 27 illustrates the main events that occur in the different levels when an application program requests data.

(2) Data Dictionary

The data dictionary is the database in which descriptions of the operational database are kept. The database administrator function has responsibility for creating and maintaining the data dictionary, enabling it to be used as an effective tool for project control, communication and standards in the design, implementation and operational phases of the database's life cycle.

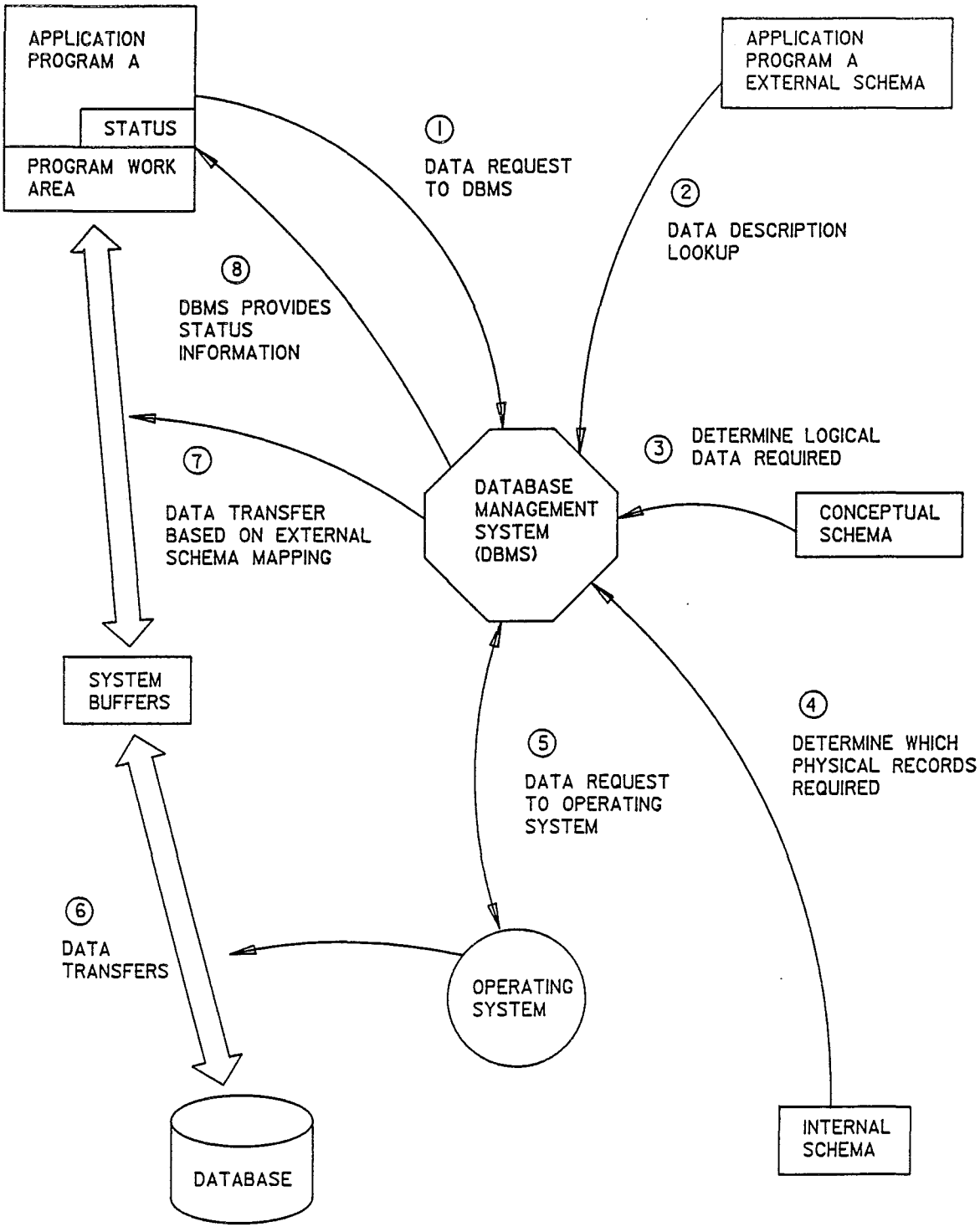


Figure 27. Main Events When an Application Program Requests Data from a DBMS (Source: Martin,1977)

In particular, a comprehensive dictionary contains the following:

- The set of consistent schemas (external, conceptual, logical and internal) are included in text form and in compiled form;
- A description of itself.

The dictionary should support different versions of the schemas and data elements each with its own status (for example , proposed, or approved). The integration of the dictionary with applications and the DBMS should enable any changes in the applications to be reflected automatically in the program descriptions in the dictionary, and any changes in the database to be the result of changes in the database description in the dictionary. Ideally the binding between external schemas and internal schemas should be done at execution time when the database description and program description are taken from the data dictionary.

As a database, the data dictionary is subject to similar design criteria, operational practices and uses as any database. For example, descriptions of host language programs, external schema, and listings of departments using data elements are useful in providing an indication of the effect of change to the data model on programs and transactions. Both ad hoc and formalised requests should be easily made but subject to controlled access whether they are made through a host language or interactive users language.

The data dictionary should provide information for cost/efficiency studies of the system by an auditing function. It should also enable (from the various descriptions) the generation of the data descriptions for host language programs, external views and standard access (input/output) modules.

Atre (1980) discusses the data dictionary.

(3) Introduction To Normalisation

Normalisation theory resulted from Codd's and others' observations that certain relations have better properties in an inserting, deleting, updating environment than other relations. The formalised ideas in it are a useful aid in the design of logical structures for databases, but are not laws and there may be good reasons why the logical structures are not in the "more desirable" normal forms, namely third, fourth, fifth normal forms.

As the wording in this section is that of relational databases, a brief introduction to this terminology is called for before proceeding with normalisation. This in no way implies that normalisation should or can be applied only to relational databases, rather that it gives it a formalised basis.

(a) Relational Data Structures (refer Date, 1981, chapt. 4). Given a collection of sets D1, D2, ..., Dn (not necessarily distinct), the Cartesian product of these n sets, written D1xD2x...xDn, is the set of all possible ordered n-tuples <d1, d2, ..., dn> such that d1 belongs to D1, d2 belongs to D2, ..., dn belongs to Dn. For example, figure 28 shows the Cartesian product of two sets MAJOR_P# and MINOR_P#.

R is a relation on the sets D1, D2, ..., Dn if it is a time varying subset of the Cartesian product D1xD2x..xDn. The value of n is the degree of R. Sets D1, D2 ..., Dn are the domains of R.

Relations of degree one are said to be unary, relations of degree two are binary, ..., and relations of degree n are n-ary.

Strictly speaking there is no ordering among the tuples, because a relation is a set, but in practice there is some particular unchanging

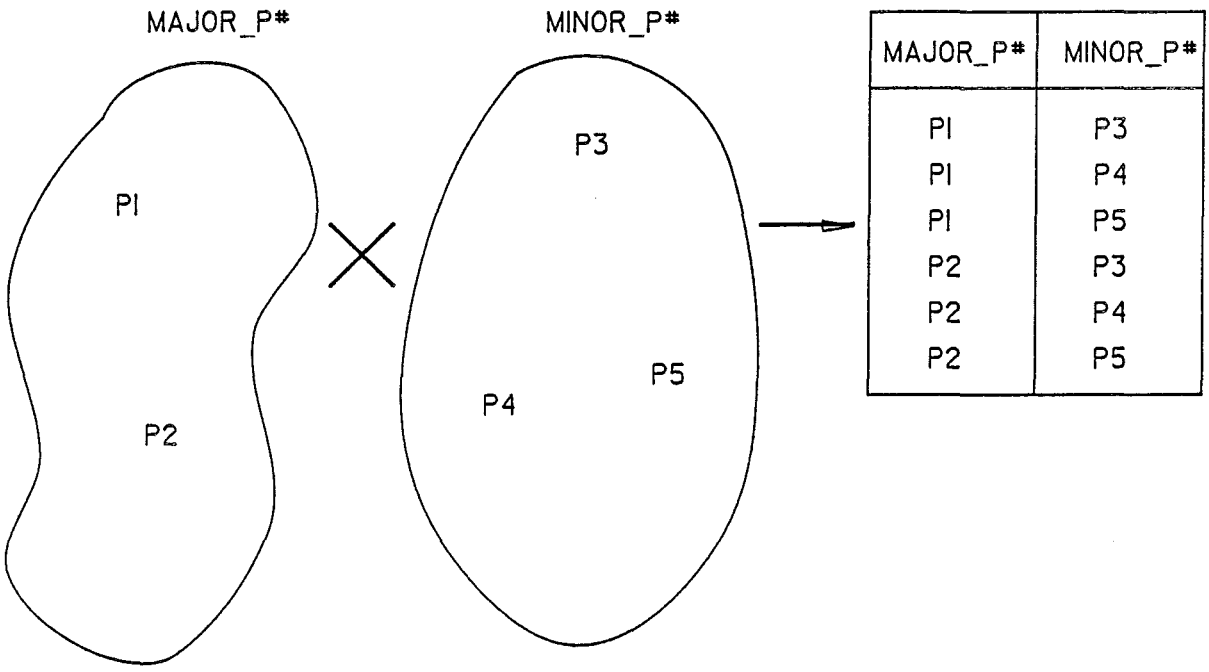


Figure 28. An Example of a Cartesian Product

order (the user does not care what the ordering is) so that sequential retrievals of more than one tuple can work.

An attribute represents the use of a domain within a relation; they characterise the relation R. In figure 29 there is a relation with three attributes but only two distinct domains. The meaning of a tuple of the relation COMPONENT is that the major part includes the minor part in the indicated quantity as an immediate component.

Codd has shown that there is no fundamental advantage in having domains composed of other relations and, instead, proposed that relations be built exclusively on domains of elementary values, for example, integers, character strings and so on. Thus a normalised relation is that in which each attribute value in each tuple is atomic (i.e. nondecomposable so far as the system is concerned). A simple domain is one in which all elements are atomic. A trivial example of normalisation is shown in figure 30. The possibility of null values (special values representing "unknown" or "inapplicable") is allowed.

One or more domains whose values uniquely identify the tuples of a relation is called a candidate key. For example in figure 29 the combination of (MAJOR__P#, MINOR__P#) is a candidate key in the relation COMPONENT. One of the candidate keys is chosen as the primary key so as to minimise the number of key attributes, and maximise meaning to the users. In the relation COMPONENT the combination of (MAJOR__P#, MINOR__P#) is the only candidate key and therefore it must be the primary key.

The entity integrity rule states that no component of a primary key value may be null. This follows from the requirement that the primary key uniquely identifies tuples, that is, distinguishes them from each other.

COMPONENT

MAJOR_P#	MINOR_P#	QTY
P1	P3	2
P1	P4	4
P1	P5	1
P2	P3	3
P2	P4	9
P2	P5	8
P6	P1	2
P6	P2	3

Figure 29. The Relation COMPONENT

BEFORE

MAJOR_P#	PQ	
	MINOR_P#	QTY
P1	P3	2
	P4	4
	P5	1
P2	P3	3
	P4	9
	P5	8
P6	P1	2
	P2	3

AFTER

MAJOR_P#	MINOR_P#	QTY
P1	P3	2
P1	P4	4
P1	P5	1
P2	P3	3
P2	P4	9
P2	P5	8
P6	P1	2
P6	P2	3

Figure 30. An Example of Normalization

Similar considerations lead to the referential integrity rule. First a primary domain must be defined. A given domain may optionally be designated primary if and only if there exists some single-attribute primary key defined on that domain.

Referential integrity rule : Let D be a primary domain and let R1 be a relation with an attribute A that is defined on D. Then, at some given time, each value of A in R1 must be either null or equal to v, say, where v is the primary key value of some tuple in some relation R2 (R1 and R2 not necessarily distinct) with its primary key defined on D. (Note that the constraint is trivially satisfied if A is the primary key of R1.) The attribute A is called a foreign key. For example the attribute MINOR_P# of relation COMPONENT is a foreign key, since its values are values of the primary key of the PART relation.

The integrity constraints are then: the no-nulls constraint on primary keys; uniqueness constraint on primary keys and alternate keys; referential constraint on foreign keys; and any other user defined constraint. The definition of the attributes constituting the keys informs the DBMS of the dependencies between the data elements. The DBMS will then be able to enforce these dependency constraints. For a relation that is not in "Boyce/Codd Normal Form" (BCNF) or higher, additional specification would be necessary.

(b) Normal Forms (refer Date, 1981, chapt. 14). A relation is said to be in a particular normal form if it satisfies a particular set of constraints. Unconstrained construction of normalised relations can lead to semantic anomalies.

(i) First Normal Form. An unconstrained normalised relation is in first normal form (1NF):

A relation R is in first normal form if and only if all underlying domains contain atomic values only.

(ii) Functional Dependence. Before going on to introduce briefly other normal forms, the fundamental notion of functional dependence (FD) must be introduced.

Given a relation R, attribute Y of R is functionally dependent on attribute X of R if and only if, whenever two tuples of R agree on their X-value, they also agree on their Y-value.

For example, in the relation PART, attributes PNAME, DESIGNER and APPROVEDATE are each functionally dependent on attribute P#. In symbols,

PART.P# ⟶ PART.PNAME

PART.P# ⟶ PART.DESIGNER

PART.P# ⟶ PART.APPROVEDATE

or more concisely,

PART.P# ⟶ PART.(PNAME, DESIGNER, APPROVEDATE)

The statement "PART.P# ⟶ PART.PNAME" for example, is read as "attribute PART.P# functionally determines attribute PART.PNAME". Both the attribute Y and X in the definition may be composite, for example, COMPONENT.(MAJOR_P#,MINOR_P#)⟶COMPONENT.QTY. A functional dependence is also a special form of integrity constraint on the relation. It is an essential part of the meaning of the data. The fact that APPROVEDATE is functionally dependent on P#, for example, means that each part has precisely one approval date.

An attribute Y is fully functionally dependent on attribute X if it is functionally dependent on attribute X and not functionally dependent

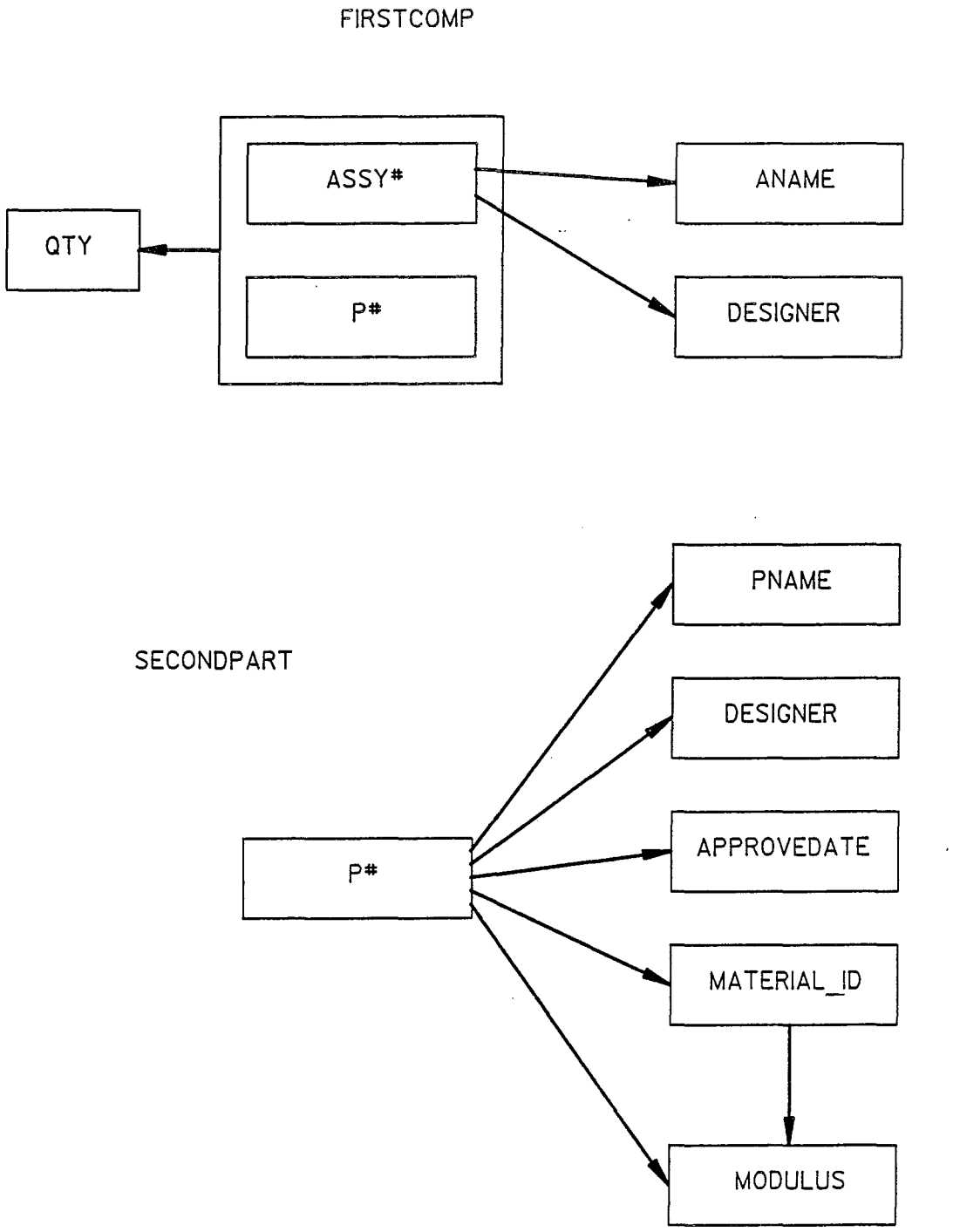


Figure 31. Functional Dependencies in the Relations FIRSTCOMP and SECONDPART

on any proper subset of X. For example, in PART, the attribute PNAME is functionally dependent on the composite attribute (P#,APPROVEDATE); however it is not fully functionally dependent on this composite attribute because it is functionally dependent on P# alone.

Functional dependencies may be represented by means of a functional dependency diagram (refer figure 31)

In figure 31, ANAME and DESIGNER are not fully functionally dependent on the primary key and, MATERIAL_ID and MODULUS are not mutually independent. It is these two facts that will lead to the update problems described by Date.

Replacing the relation FIRSTCOMP by two relations COMPONENT and ASSY solves some of these problems, that is, eliminating the nonfull functional dependencies.

(iii) Second Normal Form. A relation R is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key. (An attribute is nonkey if it does not participate in the primary key.)

Relations COMPONENT, ASSY and SECONDPART are in 2NF, but FIRSTCOMP was not (fig. 32). The replacement of the original relation by suitable projections is reversible and loses no information.

(iv) Third Normal Form. The SECONDPART structure still causes problems however. The dependency of MODULUS on P# is transitive (via MATERIAL_ID). Each P# value determines a MATERIAL_ID value and this in turn determines the MODULUS value. Again the solution to the problem is to replace the relation SECONDPART by two projections PART and MATERIAL (fig. 32). Once again the process is reversible.

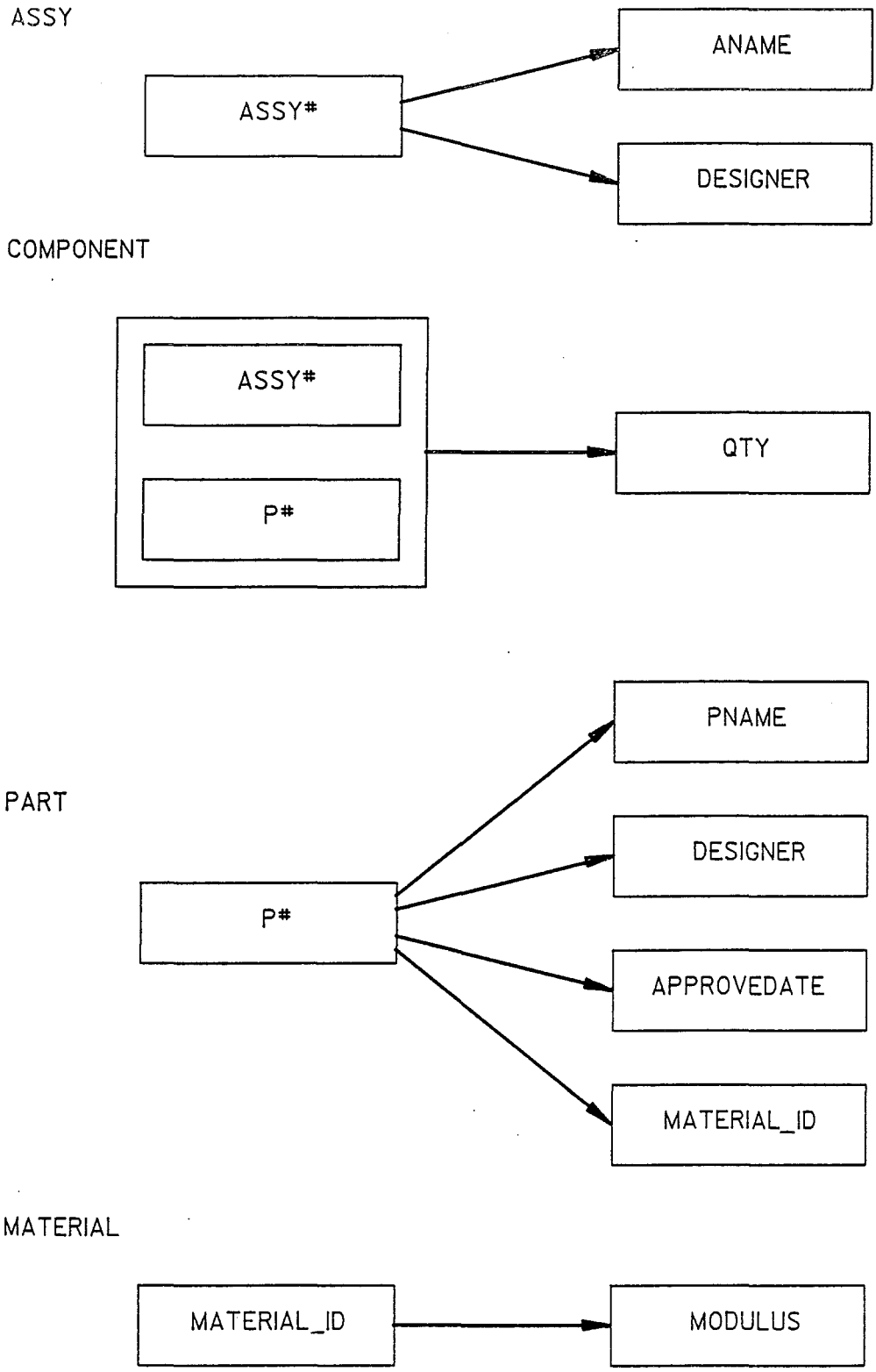


Figure 32. Functional Dependencies in the Relations PART, MATERIAL, COMPONENT and ASSY

A relation R is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

ASSY, PART, MATERIAL and COMPONENT are now all in 3NF.

(v) Boyce/Codd Normal Form. An attribute, possibly composite, on which some other attribute is fully functionally dependent is called a determinant. Then a relation R is in Boyce/Codd Normal Form (BCNF) if and only if every determinant is a candidate key. BCNF was introduced because the original 3NF definition does not satisfactorily handle the case of a relation possessing two or more composite and overlapping candidate keys. Again any relation can be decomposed in a nonloss way into an equivalent collection of BCNF relations.

Often a given relation may be decomposed in a variety of different ways. As a guideline the relation should be decomposed into independent components, so that, legal updates may be made to either component relation without regard for the other. This can be achieved by making a transitive functional dependency the inter-relational constraint, and the functional dependencies the intra-relational constraint. In other words the projections R1 and R2 of a relation R are independent if and only if:

- every functional dependency in R can be logically deduced from those in R1 and R2, and
- the common attributes of R1 and R2 form a candidate key for at least one of the pair (Rissanen, 1977).

Further normalisation is presented in Date (1981). For example, fifth normal form is regarded as the ultimate normal form with respect

to the operations projection and join, but it is less straightforward to achieve. Research continues into normalisation.

Once the data items and their dependencies have been defined, normalisation could be carried out automatically by a suitable algorithm. Thus the problems of database design lie primarily in the identification of the data elements and their semantics.

(4) Data Models

Codd introduced the concept of data model, the combination of a class of data structures and the operations allowed on the structures of the class. However the term "model" has now been applied retroactively to early data structuring methods, for example the "hierarchical model", and has come to denote only the abstract data structure class.

(a) Hierarchical Structures. Figure 33 shows a simple hierarchical view. The data is represented by a simple tree structure, with assemblies superior to parts (a one to many relationship). Examples of more complex hierarchies and discussion of commercial hierarchical systems may be found for example in Date (1981). (Date covers IBM's Information Management System.) In general the superior (or parent) data item type may have many subordinate (or child) data item types and each child may have any number of child types and so on to any number of levels, but each child type must have only one parent. Thus if a given data item occurrence, for example ASSY1, is deleted, so are all its children. For one occurrence of any given data item type there may be any number of occurrences of each of its children.

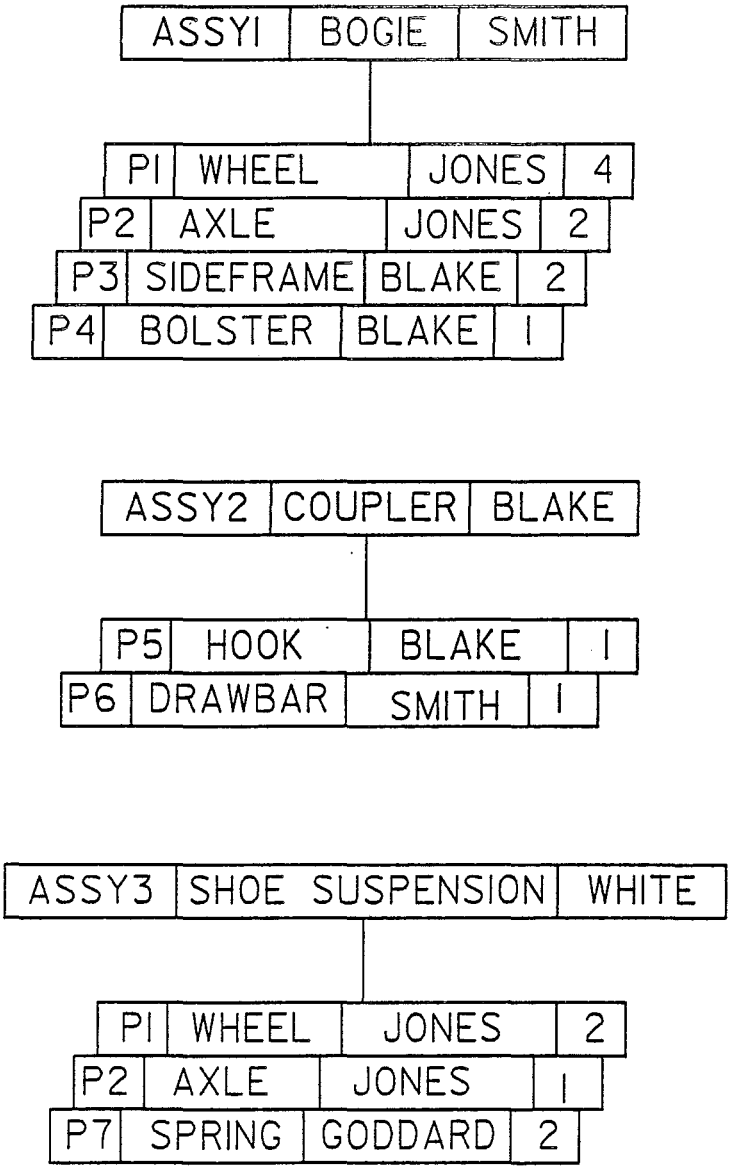


Figure 33. Sample Data in Hierarchical Form

Retrieval queries may not be symmetric in the hierarchical approach. This is a direct consequence of the asymmetric view with superior and dependent data items. Users are forced to spend more time writing, debugging and maintaining programs, because the hierarchical data structure does not reflect the intrinsic properties of the questions being asked. Update anomalies arise if the relationships are inherently many-to-many (that is, a child with more than one parent) because stored redundancies exist (Date, 1981, chapt.3).

When the data relationships are genuinely hierarchical, the hierarchical systems are of proven benefit. However even genuine hierarchical structures tend to develop into more complex many-to-many structures with time.

(b) Network Structures. In network structures, as in hierarchical structures, the data is represented by records and links. However the network approach is more general than the hierarchical because many-to-many relationships can be modelled more directly using a record to describe the association between other records (a connector) and link all of the connectors for a given record together by means of a chain, as well as other information-bearing constructs. These connectors provide access paths as well as associating the two record types involved. Figure 34 illustrates a simple network data structure.

Much of the interest in the network approach can be traced back to the work done by the Data Base Task Group of CODASYL.

The procedures for retrieval queries and update are more complicated than hierarchical, but have more symmetry. They also force the user to select a logical access path as there may be many strategies for performing operations on occurrences. This choice can be

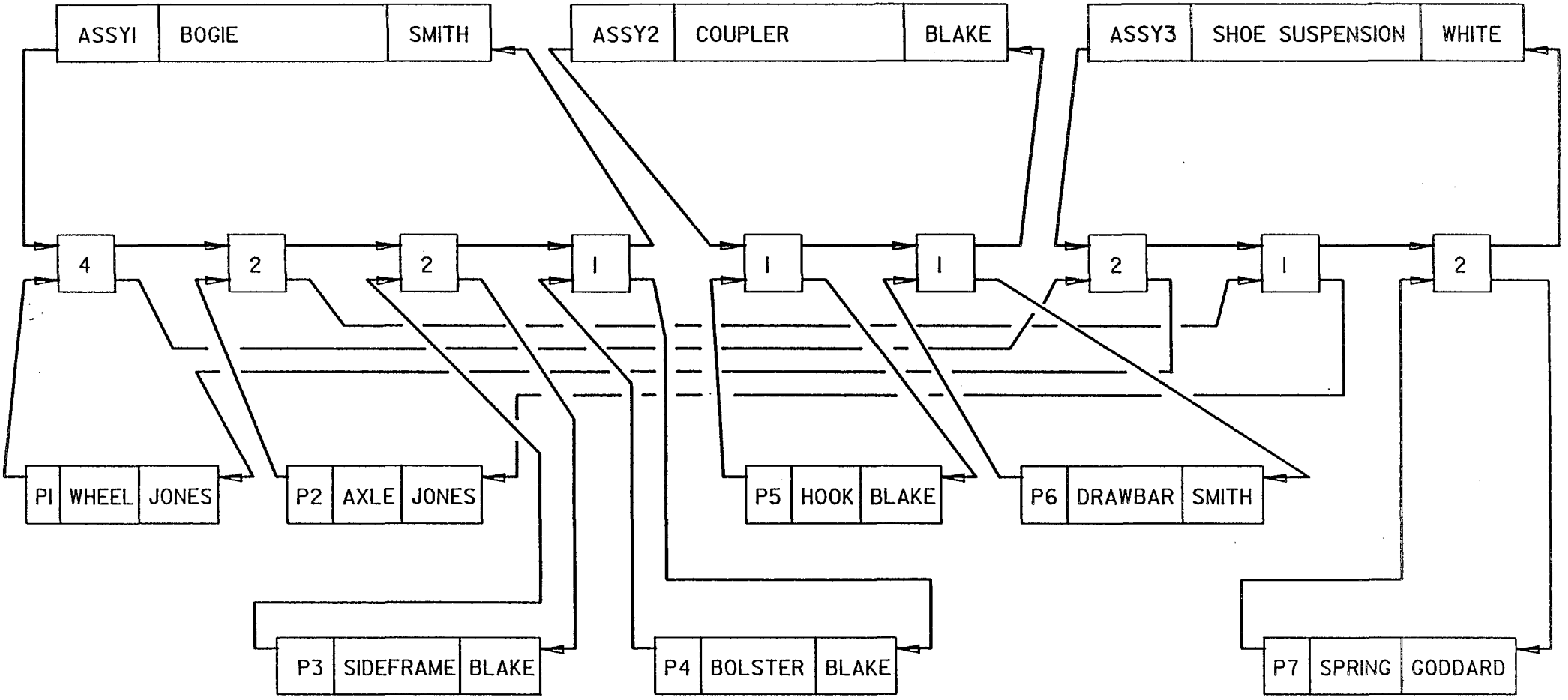


Figure 34. Sample Data in Network Form

ASSY

ASSY#	ANAME	DESIGNER
ASSY1	BOGIE	SMITH
ASSY2	COUPLER	BLAKE
ASSY3	SHOE SUSPENSION	WHITE

PART

P#	PNAME	DESIGNER
P1	WHEEL	JONES
P2	AXLE	JONES
P3	SIDEFRAME	BLAKE
P4	BOLSTER	BLAKE
P5	HOOK	BLAKE
P6	DRAWBAR	SMITH
P7	SPRING	GODDARD

COMPONENT

ASSY#	P#	QTY
ASSY1	P1	4
ASSY1	P2	2
ASSY1	P3	2
ASSY1	P4	1
ASSY2	P5	1
ASSY2	P6	1
ASSY3	P1	2
ASSY3	P2	1
ASSY3	P7	2

Figure 35. Sample Data in Relational Form

significant. The schema designer also has to choose whether to express say relationships as records and connectors or represent it in another way. Therefore the prime disadvantage of the network approach is unnecessary complexity due to the range of constructs and the operators needed to handle them, in both the data structure and the DML (Date, 1981, chapt. 3 and 28).

(c) Relational Structures. Figure 35 shows some sample data in relational form. The tables are a special case of the construct known as a relation. The rows correspond to the tuples and the columns to the attributes. The tables resemble a conventional sequential file with the rows corresponding to records of the file and columns to fields of the records. The relational approach is based on the realisation that files that obey certain constraints may be considered as mathematical relations and hence relation theory may be brought to bear on practical problems arising from the use of data in such files.

All relationships between entities are represented in the same way as entities (that is, tuples in relations), unlike in the hierarchical and network approaches which use links in certain relationships. In relational data structures associations between tuples are represented solely by data values in columns drawn from a common domain. For example, the relations PART and COMPONENT have a common domain - namely part numbers.

The relational structures are easy to understand. The relational model is a logical structure rather than a physical one; the various users when viewing the relational structures need not concern themselves with the complexities of the physical implementation. The simplicity of the structures and the use of entity identifiers (rather than pointers

or hardware-dictated structures) for representing entity associations are major steps forward for data independence. The details of storage structure and access strategy have been removed from the users interface.

The uniformity of the data representation (that is, the one construct the relation or the more familiar term the table) leads to uniformity in the set of operators provided to manipulate data, that is, the DML. Since the information can be represented in only one way, there need by only one operator for each of the basic functions (insert, delete, update, retrieve). This is true at both the low level of tuple-at-a-time and at the high level of set-at-a-time. For the same reason only a single set of operators is required for authorisation and integrity operators.

Because there is no positional dependency between the relations, requests do not have to reflect any preferred structure and therefore can be nonprocedural and symmetric.

(5) Operations on the Database

When an end-user performs an operation on the database, the following basic steps are involved:

- accept a command from the terminal;
- analyse the command;
- perform the required operation;
- return a message (which may include selected data values) to the terminal.

Much the same steps are involved if an application program operates on a database, but communication (perhaps via a CALL statement) will be

with the program data area. The instructions may be compiled or interpreted.

The basic functions of the Data Manipulation Language are:

- retrieve data values from the database;
- delete data values;
- insert values (make or create entity or relationship instances);
- update or modify existing entities or relationships.

The data description language should have similar operations available to specify data structures.

The above operations may be applied to tuples-at-a-time or to sets-at-a-time. Codd has postulated a relational algebra, the operators of which take one or two relations as their operands and produce a new relation as a result. In addition to the conventional set operations, the relational algebra provides such operations as SELECT, to construct a new table by taking a horizontal subset of an existing table; PROJECTION, to delete selected domains of a relation; and JOIN, to join (over some common domain) two relations into one (Codd, 1972). It is one of the strengths of the relational approach that a set level DML (which is simple and powerful), such as relational algebra, can be readily defined.

The concept of a relational calculus (that is, an applied predicate calculus specifically tailored to relational databases) was also first proposed by Codd. The distinguishing feature of this tuple oriented relational calculus is the notion of the tuple variable - a variable that ranges over some named relation. (refer Date, 1981, chapt. 13 and Codd, 1972 for further details).

Codd has also proposed that the relational calculus be used as the standard against which other DML's could be measured for completeness. He defined relational completeness as : "a language is relationally complete if, given any finite collection of relations R1, R2, ..., Rn in simple normal form, the expressions of the language permit definition of any relation definable from R1, R2,..., Rn by expressions of the relational calculus." (Codd, 1972).

Relational completeness provides a tool for measuring the selective power of a database language.

More recently a relational calculus, the domain calculus, has been proposed (Lacroix and Pirotte, 1977) in which variables range over the underlying domains instead of over relations. However Ullman (1979) has shown that expressions in domain or tuple calculus or relational algebra can be converted to equivalent expressions in each of the other languages, thus the three languages are all equivalent to each other in their selective power.

Then set-level relational DML's have (Date, 1981, p.139-140) :

- Simplicity. Problems can be expressed more easily and concisely than in low level languages often using a single statement. Simplicity will result in increased program production and maintenance.

- Completeness. Relational completeness means that, for most queries, the user need never resort to loops or branching.

- Non-procedurality. The user specifies only what is wanted, a statement of intent, not a procedure for obtaining that data. This makes search optimisation possible and helps in the implementation of authorisation checking.

- Data independence. The set-level DML's can provide total

physical data independence - statements include no reference to explicit access paths - and a certain amount of logical data independence.

- Ease of extension. The retrieval power can be easily extended by system or user defined built-in functions or routines (see for example MIMER/QL procedures).

- Support for higher-level languages. It may be preferable to provide an end-user language that is tailored to a particular application area, that is, a Problem Oriented Language (POL). The set-level DML's can be used as an internal language - as an intermediate step in translation - because they provide a common core of features that will be required in some shape or form in all POLs.

(6) Relational Database Model Definition

Date (1981, p.214) defines the relational database model to consist of two principal components:

- the relational data structure; and
- the relational algebra.

Furthermore a database system could be called fully relational if it supports:

- relational databases (including the concepts of domain and key and the two integrity rules); and
- a language that is at least as powerful as the relational algebra.

A system supporting relational databases but with a language less powerful than the algebra could be called semirelational. However relational definitions continue to evolve as research results in additions and clarifications to the original definition.

4. REFERENCES

ATRE, S. (1980) Data Base : structured techniques for design, performance, and management. New York, John Wiley and Sons. 442p.

BRITISH STANDARDS INSTITUTION. (1986) Specification for Single-hit Decision Tables. (BS 5487:1986).

CHOMSKY, N. (1959) On Certain Formal Properties of Grammars. Information and Control, 2(2): 137-167.

CHVALOVSKY, V. (1983) Decision Tables. Software Practice and Experience, 13:423-429.

CODD, E.F. (1972) Relational Completeness of Data Base Sublanguages. In Data Base Systems, Courant Computer Science Symposia Series, Vol. 6. Englewood Cliffs, New Jersey, Prentice-Hall.

DATE, C.J. (1981) An Introduction to Database Systems. 3d. ed. Reading, Massachusetts, Addison-Wesley . 574 p.

DAVIE, A.J.T. and MORRISON, R. (1981) Recursive Descent Compiling. Chichester, England, Ellis Horwood. 195p.

FENVES, S.J. (1976) Formal Representation of Design Requirements. In SAUL, W.E. and PEYROT, A.H. ed. Methods of Structural Analysis. New York, ASCE. p.739-755.

GOEL, S.K. and FENVES, S.J. (1971) Computer-Aided Processing of Design Specifications. Journal of the Structural Division, ASCE, 97 (ST1) : 463-479.

HUNT, A.A. (1983) Computer Aided Design of the Body Structure of

Bogie Railway Wagons. Christchurch, University of Canterbury. (Final year project report : B.E. : Mechanical Engineering).

KONOPASEK, M. and PAPACONSTADOPOULOS, C. (1978) The Question Answering System on Mathematical Models (QAS) : description of the language. Computer Languages, 3 : 145-155.

LACROIX, M. and PIROTTE, A. (1977) Domain-oriented Relational Languages. Proc. 3d. International Conference on Very Large Data Bases.

McDANIEL, H. (1968) An Introduction to Decision Logic Tables. New York, John Wiley & Sons.

MARTIN, J. (1977) Computer Data Base Organisation. 2d. ed. Englewood Cliffs, New Jersey, Prentice Hall. 713p.

POLLOCK, S.L. (1971) Decision Tables. New York, Wiley - Interscience.

RISSANEN, J. (1977) Independent Components of Relations. ACM Transactions on Database Systems, 2(4): 317-325.

TREMBLAY, J.B. and SORENSON, P.G. (1982a) An Implementation Guide to Compiler Writing. New York, McGraw-Hill. 259p.

TREMBLAY, J.B. and SORENSON, P.G. (1982b) The Theory and Practice of Compiler Writing. New York, McGraw-Hill. 600p.

ULLMAN, J.E. (1979) Principles of Database Systems. Washington, D.C., Computer Science Press.

CHAPTER X

A FUNCTIONAL SPECIFICATION FOR THE INTEGRATED COMPUTER-AIDED WAGON DESIGN SYSTEM

The preceding chapters have described the processes and tasks involved in wagon design. The ill-defined nature of design was discussed, which for the present requires the essential contribution of the wagon designer. Design was described as an incremental activity which accumulates knowledge as it proceeds from its initial state to its goal state. The goal state partially defined by constraints and objectives typically contains many acceptable solutions with stylistic variations. It was observed that the wagon design system underwent continual adaptation. A wide range of models were observed to be used in wagon design; models that approximate the real world and represent the relationships amongst some attributes of the design. Some of these attributes may be varied as alternatives are pursued. Through the use of numerous models and evaluation of many alternatives a vast amount of information describing the design is amassed as the design proceeds. How these models and the design data are used in the synthesis of alternatives, their evaluation, and the direction of the process or design path is the control aspect or "method" of design.

The preceding chapters have also suggested areas in which computers might beneficially be applied to wagon design and they also contain brief descriptions of how computers have been applied to wagon design. These areas include: more automated information processing in clerical and arithmetical tasks; more assistance to the human information

processing system; a large store for controlled sharing between users and programs of precise descriptions of design alternatives, and the design path; more "self-knowledge" on where to find information and how to use it; and more management control over design procedures, and documentation, over component variations, and over synchronization of processes and release of information.

1. REQUIREMENTS FOR A COMPUTER-AIDED WAGON DESIGN SYSTEM

The computer-aided wagon design system can be defined as those elements of the wagon design system which are computer-based and the ancillary equipment and procedures which directly support these elements.

The design of a computer-aided design system is subject to a number of objectives and constraints. With the preceding review of wagon design in mind, the following constitutes a set of criteria by which the candidate systems may be judged.

(1) Portability

The programmed part of the computer-aided wagon design system must succeed in substantially reducing the cost of:

(i) Allowing users to use the system on a variety of different computer systems. Transferring the system from one computer system to another should make no difference to the semantic meaning and syntax of any command or to the responses to these commands. On all computer systems there must be a one-to-one relationship between syntactic form and semantic meaning.

(ii) Transferring the system between different computer systems. Reduced cost in transferring between computer systems will amongst other things extend the life of the system by making it easier to transfer to a newer system that has more functions and is more cost effective.

Total portability may only be achieved where it is possible to implement all of the system by compilation of the source code and input of the data files. As this is not always possible, it must be easy to locate those parts of the system which are dependent on the target computer system. It should be a rare occurrence that it is not possible to implement a feature of the system.

Cost reduction requires the maximum in terms of portability of the system, that is, a high degree of independence of the programs from any particular machine and its software. This insulation from the environment of the system may be achieved by using a high level language compiler as a buffer, a widely implemented operating system, and device independence and communication standards. Modern techniques will provide a design which will not be locked into the past.

(2) Ease of Use

Normally user productivity is one of the most important criteria. An effective system is one in which the user carries out his work with minimal conscious attention to his tools and maximum effectiveness. It is free of distractions from the "real" task at hand. It also maximises the pleasure the user derives from the process and minimises psychological blocks such as user frustration, confusion, discomfort, boredom, and panic. The time spent by the user in accomplishing the tasks which the system is intended to support is directly influenced by

ease of use.

Some aspects of ease of use are:

(i) The number of operators defined for the associated CAD system objects should be small.

(ii) Concise and powerful high level operators should be available. The operators must be precise but close to the imprecise "operators" of natural language. Irrelevant decisions and detail should be removed. Access to data should be gained in simple fashion.

(iii) The system should be flexible with regard to what can be done, how it can be specified and when it can be done. The value system of the computer-aided wagon-design- system designer should not be imposed on the user. The user should be able to use the system in ways not foreseen by the system designer. So long as the user is making logical use of the system, he should be free to wander around and focus on data and functions. The system should not artificially constrain how it is used.

(iv) Short-term memory overload due to such things as visual clutter result in poor performance, frustration, fatigue, and errors. High short-term memory load results when the user is obliged to retain unprompted knowledge of task elements over the duration of the task.

(v) The design of a task should take into account the context of the task among temporally adjacent tasks and the existence of global patterns of task sequencing.

The effort in using a number of application programs (or groups of

functions) must be minimised. Activity analyses show that not only does the wagon designer spend time doing different things that could be classified as design, but significant portions of his time is spent in other classes of work.

The relationship between tasks and the technique for performing those tasks on the system is strongly influenced by user experience and knowledge. A knowledgeable user requires a wider range of facilities and finer, more precise techniques than an inexperienced user, before he will regard the system as efficient and satisfying.

(3) Ease of Understanding

Comprehensibility will make the system capabilities easier and more efficient to learn. It should be possible with the interface language to construct an easily understandable statement of the tasks to be performed by the system. It should be clear to the user what data is available. Facilities will need to be easy to use independent of when the user last accessed the facilities. The experienced user can tolerate a much higher "memory load" with fewer prompting features. Indeed, a system that works well for the inexperienced user can be unproductive, crude and frustrating to the experienced user. Together with the experience of the user, learning and recall time determine the user's skill level, and thereby affect task time. Comprehensibility of the system will also affect training costs.

Some aspects of ease of understanding are:

(i) The number of distinct basic constructs should be kept to a manageable and convenient size. Long-term memory is used to recall the

details for using the system. The long-term memory load can be minimised by using a small number of steps and a small amount of key information, otherwise learning and recall may be slow.

(ii) Distinct concepts should be cleanly separated, so that the construct can serve only one purpose in a given situation.

(iii) Naturalness should be preserved. When the user learns how to use the system, he acquires and organises information concerning its use. If the information fits into categories or concepts the user already understands, then learning can proceed rapidly; if not, learning is slower.

(iv) Prompting the actions as well as the data are useful ways to reduce long-term memory load. Assisting the user in other ways such as explanations of functions also adds to comprehensibility.

(4) Predictable Behaviour

Consistent and logical interaction techniques, system responses, and system actions will assist learning whereas uncertainty will lead to fatigue, errors and dissatisfaction. If the system has regular patterns to its behaviour, long-term memory load will be reduced. A theoretical base will ensure predictable behaviour and awareness of limitations. It will mean that if the system is used in accord with theory, ambiguity and paradox will not occur, and that the users can build an intuitive model of its behaviour with no exceptional conditions.

(5) Reliability

Reliability is the likelihood that the system will function in an acceptable manner. Robustness and consistency are aspects of reliable

functioning. All other attributes suffer if the software is unreliable. The software must undergo systematic testing to achieve reliability. Reliability of data (the probability of the data being free of system detected errors when accessed) should be user definable.

(6) Performance

The response time should be appropriate to the complexity of the processing requested, that is, for simple actions and small changes the response time is quick (typically under a second for trivial interactive requests). For longer processing times there should be feedback of anticipated completion time and progress status messages. There should be dependable performance so as not to artificially constrain or destroy concentration of the user. Unpredictable performance will frustrate the user. Monitoring and design aids should be available to assist in the design and implementation of improvements.

(7) Availability

The system facilities and data should be available whenever the users need them. Availability of facilities or data should be defined by the user in terms of mean time to access or management cost.

(8) Development Cost

The cost of producing the computer-aided wagon design system should be minimised. The majority of potential users will not change to the system unless there are sound "cost-benefit" reasons for doing so. As a consequence of this, no viable proposal will involve the rewriting or modification of significant amounts of existing software (for example,

operating systems, text editors, compilers, database management systems, and application programs) because of the enormous cost involved. The better of these software products may be treated as independent components of the system or modules and purchased or leased for the system.

The investigations into wagon design have shown that there is a vast number of possible design functions and tasks that could be programmed. These applications vary in complexity from the relatively simple formula manipulations to large structural or dynamical analyses. It is reasonable to assume that not all the applications will be programmed within NZR (or for that matter within any single organisation) and further that they will not be developed within a short time span.

Cost effective acquisition of the system can be achieved by ensuring that there is no duplication of software function. Each function should only be developed once. Programmer productivity should be maximised by making program development cheaper and faster.

(9) Functional Scope

All wagon design activities (some of which are outlined in chapters II to VII) that can show a surplus of benefit over cost of implementation (the level of which will be determined by NZR management) should be programmed on the system. Generally procedures will be computerised only when their benefits for each time they are used multiplied by the number of times they will be used outweighs their implementation cost. Aspects of design models and methods were discussed in chapter VIII.

(10) Extensibility

Extensibility is the degree to which the system accommodates cost effective and timely change in order to implement a new requirement or a modified requirement. In a changing hardware and software environment, modification and implementation of new developments should be cheaper and faster. It should be easy to create and implement new language constructs. Existing programs and logical data structures should not have to be redone when changes are made. The system should be able to grow and change without interfering with established ways of using data or program functions. New applications should, where possible, be met with existing data structures rather than create new files for the same data.

The longevity of the system will be dictated in part by the ability to react to changes. The system should cover the present needs of the users and, as far as practicable, the needs of the future. Advances in analytical modelling of wagons and in solution techniques will require modification or replacement of the existing application programs. Operating systems will improve, internal memory will become bigger and faster, communication with the computer will approach "natural" inter-human communication, and so forth. In the world of computing one of the few things one can be certain of is that change will occur. The system will need to be adaptable if it is to survive.

(11) Efficiency

The system should efficiently use computing resources such as processor time and disk storage space. Traditionally this is an

important issue, but with increasing power, falling hardware costs, and a useful system life spanning many years the other issues are of at least equal importance. Modern computing tools allow memory, secondary storage space, execution speed and other measures to be monitored.

(12) Maintainability

Ease of maintenance of the system must be emphasized in order that the system can be kept functional and faults corrected in a cost effective and timely manner. Comprehensibility is an important attribute of the system which will aid maintenance. Overall economics will determine when it is no longer necessary to maintain aspects of the system.

(13) Completeness Of Design Description

Again any aspect of the design description that when stored can show an overall benefit should be part of the computerised system. Aspects of the design description include its temporal evolutionary nature; differing levels of detail in design models; the different application program and interactive uses made of the design description; the tentative nature of design alternatives; and backtracking and analysis of the design path. (Further discussion on the design description is contained in chapter VIII.)

(14) Controlled Access

Unauthorised access to data and programs should be prevented. Access controls to allow sharing and/or modification of data and programs where deemed necessary should be available. The same data or

program could have access restricted in different ways for different users. Access controls may operate at the attribute, instance, relation or database level. The definition of the "owner" is also required. Records may be kept of attempted security violations.

Data must be protected from people who might deliberately update them with false information, or deliberately misuse material. Access controls can also be used to control release of information between design groups. For example, after approval, information from the structural designers can be released to the brake system designers. Access controls can also provide privacy between individual users.

Central to the issue of access control is the ability of the system to identify one user or group of users from other users. Users must be positively identified.

The individual users or groups of users must have the appropriate authority delegated from the wagon design system management before the user(s) can decide when, how and to what extent information about themselves and their activities is accessible to others. The user will want to be able to specify the amount of protection he requires.

(15) Integrity of the Design Description

The data should be consistent in meaning, accurate and current; data should be free of errors when accessed. The system should prevent incorrect entry of data and protect against accidental loss of information. Once correct data (and programs) are in the system, hardware or software malfunction, or user actions should not be able to corrupt it. The data should be protected from fire and other forms of

physical destruction. It should be quickly reconstructable from system failures without loss of transactions. It should also be possible to produce a copy of the information as it was at a particular date and time, and to determine who has changed or referenced the information. The data should also be auditable. The system should avoid having different instances of the same data item that have different values. Invalid inference should not be able to be made from the data in the system.

(16) Communication of Information

The cost of communicating and sharing data with elements within the computer-aided wagon design system and with elements in its environment should be minimised. Easy and transparent network communication facilities, if required are assumed to be part of the hardware and computer systems software. Timeliness and reliability are other attributes of communicating data.

(17) Concurrent Use

The system should offer concurrent access to a number of users at the same time. The degree of concurrent access (for example, concurrent access to the same data element instance or concurrent access to the same logical data structure and not to the data element instance) will be a trade-off between the increased cost of detailed concurrent sharing and the risk of the user/program being delayed because the user/program cannot immediately access the information it desires when the information is being used by another user/program.

(18) Allocation and Recording of Resource Usage

The use of system resources, data, and facilities by users, projects, programs, and so forth should be logged and facilities should be available for reporting on resource usage. Resource usage information is used in the wagon design systems performance improvement function.

Facilities are required so that management can allocate resources amongst the users and set budget limits. All users require facilities to fix maximum resource usage for sessions and commands.

(19) Managerial Control

Wagon design system management should be able to exert control over the design procedures used, design documentation and components used. Standards can then be enforced.

(20) Centralised Control

Some form of centralised control over the system is needed in order to design and implement a system which achieves balance among the preceding criteria according to the wagon design system's value system. As with most design problems the requirements are often conflicting, for example, simplicity versus power; portability versus new hardware functions; ease of use for novices versus convenience for fluent users; minimised storage costs versus fast user response; fast searches for data versus flexible searches and so on.

2. FEATURES OF A COMPUTER-AIDED WAGON DESIGN SYSTEM

With regard to the requirements stated above some features of the ideal system are now outlined.

(1) Documentation

Detailed documentation is a prerequisite to achieving the overall objectives. Documentation should contain all the basic documents of a computer system necessary for application by the user, and for maintenance, development, training and installation activities. Further details on guidelines for documentation are available in the literature and where computer-aided design is concerned from organisations such as The Association for Computer-aided Design Inc. (ACADS).

(2) Modular Design

Modular design will allow the capability of any specific module to be enhanced without affecting other modules. Modular design involves the division of the program logic into coherent, self-contained units and structuring the data flow between the modules to achieve a simple and clear interaction between modules. Modularity facilitates addition and testing of various techniques that perform the same task. Rajan and Bhatti (1983) have presented a modular approach to finite element based optimisation. Further details are available in the literature on "software engineering".

(3) Software Engineering Tools

Software engineering attempts to build in the desired qualities

right from the start of the system life cycle as experience has shown that faults become more expensive to eliminate the later in the development process they are discovered. There are a number of methodologies, techniques and principles which are claimed to aid in the attainment of desirable qualities. There are also computer-based systems on the market for systematic documentation, testing, implementation, design, and maintenance. These tools offer a disciplined approach that automates some tasks and standardises some of these activities.

At minimum, the system developers will require high level interactive language debuggers, debugging aids, compilers and code linkers, a powerful interactive text editor, code library management facilities and a data dictionary.

(4) Encoding Language

The encoding language in which the system is written must be in standard form and not a computer system dialect in order to insulate the system from the peculiarities of any computer system.

The encoding language is an important issue as it contains the power with which the programmer can access computational tools including data structures, graphics, and command language definition. Lack of facilities within the standard encoding language will necessitate use of machine dependent functions.

Good application of a language leaves the logic structure of the algorithm clearly visible. Readability is also enhanced by the use of meaningful variable names, subprogram names, keywords, and so on. By making the program easier to understand, it is easier to debug, maintain

and extend.

The encoding language should be available on a wide range of computer systems with a large library of functions and subprograms. The object code should be efficient and reliable. Flexible input and output facilities are also desirable.

(5) Integration

There are three basic strategies for the use of computer-aids in practice; discrete application systems, interfaced systems, and integrated systems.

(a) Discrete Application Systems. The most straightforward way to introduce computer-aided design techniques in practice is to replace and augment certain manual procedures with discrete application systems which perform the same functions. This has been the predominant approach in wagon design.

The level of integration is low with this approach; perhaps all application systems work on the same computer system. The operating system of the machine is then used as the integrating mechanism through its file utilities, resource accounting and user security facilities, interactive command language, and so forth. The system text editor can be used to prepare input data and the output can be examined on terminals or printed out. The output file can not usually be used as an input file unless manual editing takes place. Because each application system is a distinct entity, each developer has often developed his own conventions for user interaction, program structure and data format.

The principal advantages of this strategy may be summarised as follows:

(i) Such application systems are widely available;

(ii) Costs of development are usually relatively low making it easier to cover them with short-term cost savings over conventional methods;

(iii) Application systems can be developed and implemented relatively quickly;

(iv) Application systems can be introduced incrementally without substantial disruption of existing systems enabling the recovery of development and implementation costs as the systems are introduced;

(v) Computing resource usage can be optimised for each application system.

However such an approach is often resource wasting, difficult for wagon designers to understand (as opposed to system developers), places unnecessary constraints on the wagon designer and can suffer from data integrity problems. As the number of individual systems increase, the functions of different systems may overlap or complement each other and it may be necessary to exchange data between related programs. The error prone manual data preparation and interpretation steps involve substantial time and effort and these steps may have to be repeated for each system. The redundancy inherent in the data files can lead to inconsistencies when the copies are manually maintained. In different systems, methods of interaction with the user can take dramatically different forms offering few opportunities for reduction in training times or easy transfer of users between modules. Many different interfaces increase maintenance and development costs. Many of the functions in operating systems are not required by the wagon designer whereas other desired functions may only be available indirectly through

the application systems or through a number of complex operating system commands. Redundancy in the programming of the discrete systems can be effort wasted in development and maintenance activities, and system qualities may vary considerably.

(b) Automated Transfer of Data. Whereas in the first approach algorithms were reusable, in this strategy repeated use is made of data for the different computations. Special mapping programs can be written to take input and/or output data from one application system and re-express it in a form suitable for input to another system. While this approach can be useful, mapping programs tend to be difficult to implement, and the resultant interlinked system of programs is often cumbersome and inflexible. There can also be information "degradation" as information and integrity is lost in the translation process. Numerous translation formats and programs may be necessary, the execution of which can consume a lot of computing resources. Unless there are special systems managing integrity by performing functions such as transaction and redundancy management, the relationships between different versions of sets of data and update propagation will be difficult to maintain. There will still be an independent entry to each application system. Data access will be inflexible and data will be time consuming to search.

(c) Integrated Systems. A comprehensive integrated system, rationally and systematically designed with integral sharing and communication of data will not have these problems.

In such a system there is a conceptual single database in which

data need only be entered once when it is required for executing a command. For subsequent commands the correct version of this data is extracted from the database. The data would be efficiently and effectively communicated to the other application functions exactly as required with no loss of meaning. Quick, efficient and concurrent access to data and data manipulating functions is provided through a single consistent interface. This interface has features for users to manipulate the design description, interrogate the database, produce the required graphical and text reports, and call for application functions as required. For data that need only be available during a session, run-time data structures provide the integration. Otherwise the results of data manipulations are stored together with their relationships to other data.

The single entry to any part of the integrated system would provide user security facilities and logging of system use as well as allowing the user to switch from one application function to another with a single command, bypassing the process of saving results, exiting a program, and entering another. For a single user, multi-tasking would allow a number of application functions to run simultaneously.

In the common user interface the command structure is identical; the same word means the same things; menus are driven in a compatible way; prompts are consistent in their meaning and style; error messages and help systems are compatible. Such an interface reduces errors, is convenient and simple and accelerates learning.

Rationalisation implies some level of centralised control or co-operation. Some of the advantages that accrue from centralised control (or co-operation/agreement) of data are (Date, 1981):

(i) Redundancy can be reduced. Master files are used in conventional non-database systems to maintain continuity between program runs. These master files are often designed and maintained independently of one another resulting in common data items appearing in different master files - a waste of storage space. Centralised control can consolidate the separate files to give controlled redundancy. This can reduce or eliminate inconsistencies inherent in maintaining multiple copies of the data. Redundancy can be controlled by representing the fact by a single entry (that is, removed redundancy) or by ensuring that any change made to any of the multiple entries is made automatically to the others (propagating the operations of creation, modification and deletion). Controlled redundancy also adds comprehensibility. Controlled redundancy may be introduced to give adequate response to enquiries.

(ii) Data can be shared. This means that not only can existing applications share data in the database, but also new applications (interactive uses or programmed applications) can use the same stored data without having to create any new stored files. Individual pieces of data may be shared among several users (and used for different purposes). Any user or application will be concerned only with some subset of the total database, but the different subsets will overlap in many different ways. With simple formulas it is not too much trouble to remember things such as payload, wheelbase, and spring-base or to flick back a few pages to find the load/deflection /movement of a component and then to insert them in the calculations of the moment. However sharing of data in more complicated models such as finite element or dynamical analysis becomes invaluable.

(iii) New applications are possible. Data consolidation can result in the development of new applications and those requiring a much wider view of information than is feasible with fragmented master files.

(iv) Standards can be enforced. Centralised control can ensure the enforcement of standards in the representation of data. Standardizing stored data formats is desirable as an aid to data interchange between systems.

(v) Conflicting requirements can be balanced. The best database for the wagon design system as opposed to the requirements of any individual user can be chosen.

(vi) Privacy can be enforced. Access can be restricted to the proper channels and authorisation checks can be defined.

(vii) Integrity can be maintained. The database administrator can define validation procedures to be used whenever any update operation is attempted. With shared data, inaccuracies can be rapidly propagated by the programs using the data, unlike in the discrete application files approach.

The user interface and application procedures will similarly benefit from centralised control. For example, it will allow procedures to be used by many application systems thereby reducing the size, development time and cost for new applications; quality standards in documentation and modularity can be enforced; access controls can ensure integrity of the procedures and ensure privacy for an individual user's or group of users' procedures.

An integrated approach reduces operating inefficiencies by putting more effort into the design of the system than in the discrete application system approach. More effort (more money and time) is

necessary to design the integrated database; to identify and standardise the relationships and data items in the individual application files. The resulting overall scheme is more complex (although successful design will conceal this complexity).

Another potential weakness of the integrated strategy is that the wagon design system is more vulnerable to security breaches as it relies on the successful operation of its single database system.

Loss of ownership of data and consequently less responsibility for the data can lead to inaccuracies not being corrected.

An integrated database can also introduce significant performance overhead when compared to the discrete application file strategy where the files can be tailored to meet individual application requirements. The gains in overall efficiency can be to some degree traded-off against loss in efficiency for specific applications. Unless there are facilities for defining external views (which again can incur performance penalties), the users will have to possess skill in making the appropriate idealisations for their own purposes of the fully detailed design description.

(d) Practical Approaches to Integration. A practical approach will require the development of a system which can be applied retrospectively to existing "standalone" application programs, file management facilities, and data, as well as one in which new developments can be made. The integrated strategy does not preclude heterogeneous distributed databases and distributed processing, so long as users perceive the system to function as one.

Integration over heterogeneous distributed databases and processing

would not seem necessary for the NZR wagon design function as it is centralised at Head Office and has up till recently had very little invested in computing system. (In 1986 a two-workstation Intergraph CAD system was purchased.) However the developments in this area, which are currently being pursued, would remove restrictions on choice of supplier and facilitate integration with other computer-based systems within NZR.

The development of a complete integrated system by NZR is not regarded as a practical approach because of the vast amount of resources required and the diversionary nature of such an effort from NZR's real business.

Integration of commercial packages by NZR will also require large resources and could be dependent on the availability of access to the internals of commercial packages. A good foundation is made up of device-independent graphics and user interface development tools, geometrical modelling and development languages designed for CAD development, standard operating system and database management and data exchange software.

The ideal of a tightly integrated system may have to be compromised for some "standalone" systems where it is not feasible to integrate. Interfacing these systems with defined file formats or the techniques being developed for distributed databases and distributed processing can be applied in these situations.

(6) Mode of Operation

(a) Control Over the Design Process. A key feature of computer systems is the amount of control they offer to the user. For computer-aided design there are at least four aspects to control over the design process.

(i) Control over the design strategy maintains the wagon designer's role as overall decision maker. By assembling the commands (from those available) in the order the wagon designer desires, and by suspending and interrupting execution when desired (based on the state of the design), the wagon designer is not restricted in how he achieves a solution. Even in a computer driven design system, it is the designer who must take the final responsibility. The designer makes the final decision as to when to accept the computed results and where to proceed to next and when to reiterate.

(ii) The freedom to specify whether newly calculated data, newly entered data, or existing data (perhaps modified) is to be used in the next calculation is an aspect of control over the design process. (Assuming existing data and calculation procedures are available.)

(iii) Control over the format, type and quantity of results printed out or displayed both during execution and after completion of processing is another aspect. For different applications, the same data should be able to be accessed with different search criteria and used in different ways. Access should be flexible and the computer, not the user, should do the sorting.

(iv) Freedom to extend and modify the system to tailor it to the needs and tastes of the wagon designer and to add knowledge and extend it in ways not foreseen by the system designer is the final aspect of control discussed. Not only data but also design algorithms should be able to be modified or created. The wagon designer may wish to change or create models, methods, and variable entities to account for new

information, new problems and new ways of viewing them. Specification of changes or additions should be of a "what" nature rather than "how". In the interests of the wagon design system, design management may wish to restrict the individual wagon designer's ability to modify the system, just as the wagon designer can only use data if he has been granted access to it. For example, the design management may give permission to edit and recompile programs to those people who are classified as system developers.

(b) Modes of Operation. There are a number of modes or styles of design program operation ranging from the fully automated to the fully modular and interactive (Inglis and others, 1985).

Optimisation, or automatic design, can be somewhat inflexible to change and time consuming to program. Thus, it is most suitable when the problem has been explored and the parameters and constraints are well defined. It is difficult to apply where there are several conflicting objective functions and a large number of constraint conditions. However for well understood problems it is fast and is most likely to achieve an optimum design. There have been attempts to develop flexible optimisation-based systems, see, for example, Balling and others (1983).

Simulation techniques, which simulate a given design problem, are most appropriate for "open-ended" problems where many designs are acceptable. Two basic types of mode of operation have been used for simulation techniques.

A dialogue may be used where the computer originates each interchange presenting the user with a series of questions and progressing through the program in linear sequence, a sequence known

from experience to be expeditious for designing the type of artefact in question. Inglis and others (1985) recommended that this approach should not be used based on their experiments to measure user acceptance, fast design time, achievement of a good design, and gain of insight into the problem by the designer. However, there will be some situations, such as where the system lacks enough information to complete a command, where computer initiated interchange gives the better design.

In the second mode of simulation technique, the user originates each interchange, selecting a command from those available to specify the design task. Design proceeds by repeated modification and appraisal. Inglis found this approach is acceptable to users and provides greater insight for the user into the problem. In interactive dialogue, the user communicates directly with the computer, when the computer system recognises an error in the user's input, it tells him so immediately. The user can then correct the input while the purpose of it is still fresh in his mind and avoid repeating the mistake in subsequent input. Alternatively if there appears to be something wrong with the computer's execution of the command, or if the user wishes to suspend or abort the execution as the user monitors progress, then he can interrupt the task and edit the command and restart the task if desired without any loss of information. This type of interaction permits new applications such as the use of less deterministic algorithms. Many of the widely accepted computer-aided draughting systems provide these run-time control features and application of these interaction techniques to the rest of the design process can be expected to have similar success.

The best mode of operation appears to be an interactive user initiated dialogue with optimisation and programmed design procedures available for selection, and use of computer initiated dialogue where appropriate. In this way the overall decision making process will stay mainly a human endeavour (being the purposeful system), with the computer acting as a powerful tool. This can be achieved by subdividing the design process into a number of distinct sub-processes some of which can be automated, with the flow from one sub-process to another being guided by the designer, the communication of information from one to another being through the design description. In this way the man-machine task division can take into account the particular capabilities and inherent limitations of each, but not preclude man from performing any task he wishes. Such a scheme will more easily accommodate changes in design methods and more easily accommodate whole or part of a wagon design. Clearly the automated design of a complete wagon is impractical at present. Any non-trivial computerised design system will be incomplete for two reasons. Firstly the scope of the real world defies complete description (of the multitude of variables, decisions and calculations) and secondly the tastes of the individual designer preclude rigorous formalisation. There are few logical paths in conceptual design and it is important that the designer be allowed to explore the implications of his decisions. To determine possible courses of action requires access to and scrutiny of relevant tasks, design information and results of design checks created so far in the design. Routine output of results of protracted and repetitive calculations will swamp the designer.

3. FUNCTIONAL AREAS AND USER CHARACTERISTICS

(1) The First Level of Functional Areas

To accommodate change a flexible "system framework" will be specified using widely available modern technology of proven practicality. The system will provide for inputting, storing, and accessing a variety of design models, methods and associated design description variables in a unified and simple manner. The system will also facilitate the exploitation of new developments in computing software and hardware. A number of applications will be implemented to demonstrate this system framework.

At the first level, the system can be divided into the following functional areas.

(a) User Interface. The user interface communicates the user's requests to the programmed functions and their responses back to the user. It converts the user input into device independent messages, parses and generates code for the execution of the tasks. It therefore contains the definition of valid dialogue. It also must manage user environment tailoring (such as defaults and renaming of keywords), and identify and action user and system errors. The user specifies the problem through the user interface; the input data, the required outputs and directly or indirectly the computational methods to be applied.

Such an approach allows a common interface to a wide spectrum of applications and decouples application functions from device and interface problems.

(b) Programmed Functions. In this area the information manipulations requested implicitly or explicitly by the user are carried

out. Programming for modification and extension and execution scheduling are aspects of programmed functions. The code generated by the user interface is executed and specific routines invoked to service the user's request. A range of design models and methods will be incorporated as well as technical office and utility functions.

(c) Data Management. Data management encompasses the organisation, integrity, privacy, and manipulation of the computerised design description that supports the design process and management of the design outputs. It functions to save data between sessions, to communicate data between different design tasks and users, to document the history of the design process from its specification through its description of its physical structure to obsolescence, to record the relationships between data elements and to record the design alternatives. The integrated database serves as a non-parochial description of the many recorded attributes of a design and its design life cycle. Designers communicate with the database through application programs and through interactive inquiry and update.

(2) User Characteristics

The needs and skills of the different users form an almost infinite range; all levels of background experience, variations in the users' rate of learning, frequency with which the user comes into contact with the computer-aided wagon design system are all variables which affect the design of the system. The different users will generally want to achieve their different goals in different ways according to individual prejudices and idiosyncrasies.

While recognising that the variation in needs/skills of users is

almost continuous and almost unbounded, it is useful to group them together into a manageable number of classes in order to study their requirements. A single person may be any combination of the following three classes of user.

(a) Wagon Designer. This user has detailed knowledge of one or more areas of wagon design but has little or no experience with computing systems. He can only use the system to design wagons. He tailors the system rather than fundamentally enhancing or changing it. The system may automatically manage information storage and retrieval in a way that the user is not aware of the manipulation. Most wagon designers in NZR are described by this category.

(b) System Developer. This user develops applications for other people to use. He is experienced in creating and running programs and in designing information structures. He has access rights to change any part of the system in order to maintain and extend it.

(c) Non-wagon Design User. This user may access information in the database or the system programs, but he cannot change any aspect of the system. He may have sufficient knowledge of computing to create and run his own programs.

(3) User Activities

At the lowest level of sophistication in the wagon designer class of user, a typical session would involve: identification of the user; a request by the user for a standard set of manipulations on data supplied at the time or supplied from storage in the system (he will not be aware of how stored data is updated and used); following response from the first operation, another request for data manipulation; and so on till

termination of the session. There is no programmed branching or repeated execution at this level; communication is a series of requests. Resource and scheduling constraints are assumed to have been specified by the system developer.

At a more advanced level the wagon designer may use files external to the system data management for input, output or both. He will create and use files of text information. He should be able to refer to the file by a convenient name. Security of these files is the responsibility of the computer system's operating system. The contents of a file may be a series of commands, data or both. Other means of creating and managing series of commands may be available, and data may be more directly manipulated in the database (for example, by using the query language). The rectification of system malfunction is not the responsibility of a user at this level. He expects each task step in his command to be executed indivisibly and a report on its success or failure to be provided on the specified output device.

The system developer class of user constructs programs, specifies and develops the detailed user interface, and specifies and develops the logical information structures in the database. He will put together a number of instructions which will be executed as a single task step by the wagon designer. The parameters to the command which invokes the task together with responses from the data management system will be used to determine which sequence of instructions will actually be executed. This level of user will also be required to be able to generate messages to the wagon designer and into the session log.

4. REFERENCES

BALLING, R.J. and others. (1983) DELIGHT. STRUCT: an optimization-based computer-aided design environment for structural engineering. Computer Methods in Applied Mechanics and Engineering, 38:237-251.

DATE, C.J. (1981) An Introduction to Database Systems. 3d.ed. Reading, Massachusetts, Addison-Wesley. 574p.

INGLIS, S. and others. (1985) A Comparison of Modes for Using CAD. Computer-aided Design, 17(5):230-234.

RAJAN, S.D. and BHATTI, M.A. (1983) Data Management in FEM-based Optimisation Software. Computers and Structures, 16(1-4):317-325.

CHAPTER XI

FURTHER ASPECTS OF THE FUNCTIONAL SPECIFICATION OF THE USER INTERFACE

This chapter expands aspects of the functional specification in the user interface area. The full scope of the user interface defines everything from the concepts the user must understand down to the finer details of screen formats, interaction techniques and device characteristics. Top-down design of an interface starts with a definition of functional requirements and identification of types of users. This step also provides insight into how the capabilities of the system can best be presented to the users. The requirements definition is used as the basis for defining the language of interaction. Defining the language starts with definition of the conceptual model with which the users must deal, then the semantics of the elements of the language, the syntax, and finally assignment of the physical devices and activities - the lexical design.

This chapter deals with requirements for the user environment additional to those already presented in chapter X. The concepts and semantics for data management and programmed utility and design functions are covered in the next two chapters.

How the requirements in these other areas are implemented in the user interface is discussed in this chapter. The detailed semantic, syntactic and lexical design of a prototype user interface is presented in chapter XVI.

1. GENERAL INTERFACE REQUIREMENTS

The user-computer interface can be regarded as being composed of two languages: with the input language the user expresses his half of the dialogue; with the output language the computer communicates to the user.

(1) The Input Language

The input language can be characterised by entities, operations to be performed on these entities, and the output required by the user. The user may define to the computer such things as data, and expressions or he may command the computer to perform operations which could include retrieval of data.

Any user wishes to supply the minimum of information required to achieve the result he desires, however the requirements of an individual user will change with time. The language and communication medium that allowed all the flexibility he could envisage initially will need to change when he has a more sophisticated task to perform.

A system of defaults is required to "hide" from the user those features in which he has no current interest. These defaults should permit the user to specify the minimum of syntactic units to indicate how some particular command or definition varies from the standard or the context in which it is specified. In depth knowledge of all features should not be required in order to use the system. A simple set of basic functions should be available for the novice and advanced features could be specified when the user has gained expertise.

The user should be provided with an environment which is natural

for him to use, or else if he is forced to work in one he dislikes there will be a greater probability of deliberate or accidental errors.

(2) Output Language

Responses to the user may of three kinds: informatory, requests, and unexpected. All kinds should be described in a way that is meaningful to the user and related to the information he provides. They should be consistent (in format) and should be related to the language which was used to initiate the task.

In particular with error messages the user wants clear concise information enabling him to correct the error. The user will also expect the system to detect mistakes (according to the rules available to the system) as soon as they are made, in order to avoid wasting the user's time and the computing resources. The system should acknowledge the entered data and indicate how it is interpreted by the system. When it detects an error it should give the user the opportunity to correct his mistake rather than terminate the interaction.

It must be remembered that the requirements of an individual user may change with time. Messages which were clear and precise when he first started may later contain insufficient information.

2. CONCEPTUAL IMPLEMENTATION MODEL

(1) Interaction Diversity

There are a multitude of interaction techniques and forms of interaction. They are all designed to perform specific tasks, each implemented on some device such as tablet, keyboard, or trackball.

The physical characteristics of any terminal should be taken into account, making use of programmable function keys, touch screens, and so on to achieve maximum simplicity for the naive users. Therefore the requirements which must be met by the user interface will differ according to the interaction devices he is using.

It is also generally accepted that any one form of interaction, for example, menu-selection, is not suitable for all users. The naming of commands and keywords is another area in which there is no widely accepted scheme. What is accepted by one user, another will find confusing, ambiguous and difficult to remember. The provision of defaults is also an area which is dependent on the user, and the requirements for which will change as the user gains experience and as his use develops or changes. The layout of the display can also be relied upon to cause debate.

(2) Outline Architecture

A two stage system would seem to overcome many of the problems of providing a flexible system that can take advantage of new interaction techniques and cater for the diversity of user requirements (Hopper, 1981).

The first stage comprises the user environment and the environment tailoring facility. The user environment converts the input issued by the user into a standard command language, which has all defaults filled in, has only one syntax and has precisely defined messages for all standard responses.

The user environment is to provide user friendly interfaces to all levels and types of users. The user can vary the user environment by

issuing commands to the environment tailoring facility.

The second stage is the standard command language interpreter which accepts the device independent, standard format commands from the user environment, analyses them, generates the run-time code, passes the code to the run-time system, receives the response from the run-time system, and from this response generates the appropriate standard message. Thus the user interface is separated from the applications.

Although the specification of the standard command language and responses are given in English, a coded from of messages and commands is expected to be the normal form of communication between the user environment and the standard command language interpreter. The English command line can easily be tested, saved as an easily comprehended trace of the session and command procedures can be easily accommodated.

3. USER ENVIRONMENT TAILORING

The scope of this work does not extend to the design of interaction techniques, the reader is referred to works such as Foley and others (1984) for guidance in this area. Nor does this work cover the physical design of interaction devices, it is assumed that ergonomically designed devices are available. This subject is also treated extensively in the literature.

(1) Requirements for User Environment Tailoring

(a) It must be possible to introduce "noise" words and to cause automatic inclusion of functional words.

(b) It must be possible to re-structure the display, modify the

layout and presentation of "forms", and to define multiple views that are selective in what they display.

(c) It must be possible to change or introduce synonyms for command entities and keywords.

(d) Commands and responses should be treated in the same way if at all possible, that is, the tailoring should be consistent in both input and output.

(e) It should be possible to define a procedure with parameters to be obeyed under defined circumstances.

(f) It must be possible to forbid certain commands or options, and to suppress mention of certain attributes by applying defaults to these attributes or options.

(2) User Description

The user environment should have access to a description of the standard commands and responses, and a description of the users objects and operation facilities. It will need to received input and provide output to the appropriate devices and transform the user language into the standard language and the standard responses into responses which will be understood by the user.

The complete specification of a description for a new user will be performed mainly by authorised system developers because it is a specialist activity that may well require management and security decisions. However there are aspects of the user description, such as synonyms and the form of interaction, which do not affect the system or other users and the user may alter these aspects himself through the environment tailoring facility.

(3) Communication with the User

Whether the session is batch or interactive the language requirements are identical, the only difference being the source of input and the sink for responses. The user's session log is always a sink for responses. A session log containing a summarised copy of every command and response in a session would aid in the isolation of system errors. It would also enable decisions on maintenance and improvement to be made on actual usage rather than "guestimates". A printable session log would enable the user to study system responses at his leisure and in an edited form could possibly be used as input to another system. The information transferred to the log would be dependent on the user description. In an interactive session the user's terminal is a second sink for responses. Each task which has been initiated but not terminated (that is, suspended or concurrent) may have a different user environment providing a virtual terminal through which all communication is channelled. (In the batch case, this virtual terminal is not a physical device).

It must be possible to be able to obtain responses in varying levels of detail. For example, an error message may by default for a particular user be a brief statement, but should he wish to do so he must be able to request a more detailed level of message such as a list of acceptable syntactic elements where the syntax error occurred.

(4) Problem Oriented Language

An easy to learn and remember language for wagon designers will be one that relates in a natural way to the field of wagon design, that is,

a problem oriented language. The procedure oriented programming languages such as Algol, Cobol, Lisp and C are essentially technical in nature and they incorporate constraints on how they can be used. As such they would be an extra dimension to wagon design that the wagon designer would have to learn and remember. For example, in the area of input/output the wagon designer should not have to worry about device numbers, opening and closing devices, rewinding, decoding input, and formatting output. Only a small subset of the facilities provided by operating system command languages and programming languages would be used directly by the wagon designer, yet his primary concern is the accomplishment of the wagon design task at hand. The wagon designer is much less concerned with the details of how an operation is done than with what is to be done. Whereas operating system command languages and procedure oriented programming languages may be efficient and friendly for system developers, they are less than satisfactory for mapping the wagon designers instructions from his environment into instructions that the computer can understand.

The wagon designers' command language would be made up from the English nouns, verbs, and so on that they use everyday. Non-alphanumeric characters should be kept to a minimum. This problem oriented language is then a candidate for the standard command language.

If the position or order of command parameters is used to communicate information such as which parameter is assigned which value then a greater load is placed on the memory of the wagon designer. An alternative is to use a keyword to identify the parameter to which the value is to be assigned.

If the system provides defaults (to be confirmed or changed by the

wagon designer), then the wagon designer is free to omit parameter names and values. The designer would then only need to communicate the essential information, the system would fill in the details and ask for confirmation for important parameters. Defaults may be applied to standard items or taken from the context of the dialogue and applied to slowly varying items, for example, the value last used for that parameter type. Free format input is of course essential.

(5) Data Entry

Data entry will be a heavily used function and there should be several methods available to simplify this aspect. Graphic and data entry forms are two common methods used. Where there is bulk data, entry in the form of tables of unlimited length can be used. This method of entry uses headings, unit definition, expressions with variables, and means of generating regular patterns. For fewer variables spread sheeting offers simultaneous display of input and output. A good screen design will support short-term memory limitations. Multivalue assignment to input variables is also useful. Within a single statement it should be possible to specify as input to the programmed task a set of parameter identifiers or values, for example, a set of force values, so that the manipulation is performed for each instance in the set.

(6) Help and Documentation Responses

Detailed information about commands, their parameters and the ways in which the user's environment can be altered should be available. All the considerations that apply to the preparation of other user responses

also apply to the on-line help and documentation information. Users should not be required to read a manual for minimal use of the system.

Users may wish merely to refresh their memories about a particular command. The enquiry might be about the syntax of a command, the effect a particular option might have, or what values are permitted for a particular parameter. At other times users may wish to browse through the documentation in the hope of learning something useful. Alternatively they may wish to find out whether it is possible to perform a particular function. Novice users may wish to learn how to use the system through using the help facility as a tutorial.

The system will have to provide assistance when required in the area of design models, methods and variables. A user may wish to know about all models, methods and variables, or he may wish to know in which models does a certain variable appear. Alternatively he may require information on all the variables in a certain model. At other times information may be required on which variables are possible dependent variables when a certain solution method is applied to a model. Some of this help information could best be displayed in graphical form, for example, the relationships between the design variables.

When the system requires a particular input, a prompt string can be displayed to guide the user about the next step. An experienced user who completes a command in one go would not receive the prompts for parameters and so on, but may receive a prompt which indicates the system is ready for the next command. All prompts should be meaningful and consistent. At any state within the interaction the user should be able to request information on the available options at that state. The display of information regarding the current state of the dialogue and

processing (that is, status information) should be provided.

(7) Terminal Handling

The user should be able to define the logical way in which his terminal is to behave. Therefore in his user definition there will have to be a logical terminal specification which can be altered by the environment tailoring facility. For a particular real peripheral there will have to be a mapping defined from the logical specification to the physical peripheral features. Such a facility could control such aspects as scroll speed, cursor style, audio feedback, type ahead buffer and colour assignments.

4. STANDARD COMMAND LANGUAGE INTERPRETER

(1) Graphics

Vision remains the most widely used means of communicating with a computer. Two important classes of visual communication are text and graphics. Both classes are considered to be fundamental to the clarity and richness of the user interface. Graphics are particularly useful in simplifying and verifying input and in simplifying output review. Graphics can be used to specify complex pictures which contain a great deal of information about shape, location and connectivity. Graphical representations of data provide a means of seeing trends and spatial arrangements allowing the user to visualise the problem more easily. The cognitive and perceptual loads on the wagon designer are reduced when graphics are used. The parallel input nature of the human eye can be compared to the serial nature of speech and written word narratives;

the mental image compared with memorizing a verbal description. Interactive graphics allows the wagon designer to examine the representation and immediately modify it. Drawing and sketching is an important part in the existing support environment for the wagon designer and drawings are well established as a means of communicating and storing designs.

Further consideration of the graphics interface is beyond the scope of this thesis.

(2) Aggregation of Commands and Batch Mode

The user may wish to execute one command at a time with immediate feedback, or he may wish to execute a logically complete series of commands (a command stream) while he remains at his interactive device(s), or alternatively he may wish to execute this block in his absence (batch mode). Exactly the same facilities should be available in all modes of operation, and running in either mode should not entail modification of the aggregated commands.

The time the computer takes for each task may vary from fractions of a second to days. Clearly if the cycle time is days, little is gained by using the system interactively. If on the other hand the computation involved is trivial, the job is best performed in interactive mode, because the data preparation and queuing for batch mode will increase the cycle time many fold. Applications of batch mode in wagon design could include tasks which require a special slow device, tasks which involve more or less the same collection of commands that do not need interaction, or a task that involves a lot of computing time (which may optionally be specified at a low priority to be executed when

it is convenient for the computer system). For the system developer, testing of programs can often be achieved more systematically by running them in batch mode with files of input data. Execution of a task in batch mode means the user can use his terminal for something else in the meanwhile.

Interactive mode may be used in conjunction with batch processing. In data preparation an interactive system, particularly one with graphical capability, will enable the wagon designer to perceive and to correct errors before submitting the job for batch processing. The wagon designer may interactively combine or select results, possibly in graphical form, in order to evaluate/interpret the batch job output.

(3) Error Handling

Facilities for software error recovery are required for the system. All messages from the operating system must be intercepted, otherwise unintelligible and often irrelevant responses will be received by the wagon designer.

There is also a requirement that in the event of a system malfunction, the system developer is informed directly. The user should not have to act as an intermediary. A separate log of detected unusual events in the system could be accessed by the system developer at regular intervals or electronically mailed to him after the user has terminated his session in which the malfunction occurred.

(4) "Break-in"

A "break-in" facility is a vital part of interactive communication. A user must be able to type a few keys, preferably a

single key, at any time and the computer will stop executing the task, and depending on the type of intervention an appropriate action is then taken. Thus the operating system must at least provide facilities for "break-ins" (interrupt, polling, or some other method) and the interface to the encoding language. All possible forms of user input supported in the operating system should also have interfaces to the encoding language.

(5) "Undo" and Confirmation

The risk of performing some undesired operation, such as deleting wanted data due to inattention, should be minimised. An "undo" or clear function and/or required confirmation from the user before the operation is performed are two ways of offering a degree of forgiveness.

(6) Abbreviations

As a user becomes more familiar with the system, he should have the option of entering brief forms of commands. For users electing to type at a keyboard this allows for commands to be issued using only that portion of a command that uniquely identifies it.

(7) System Development

A high level means of defining new commands, options and parameters to be included in the system should be provided.

5. REFERENCES AND ADDITIONAL READINGS

FOLEY, J.D. and others. (1984) The Human Factors of Computer

Graphics Interaction Techniques. IEEE Computer Graphics and Applications, Nov:13-48.

HOPPER, K. ed. (1981) User-oriented Command Language: requirements and designs for a standard job control language. Heyden and Son Ltd.

KUPKA, I. and WILSING, N. (1980) Conversational Languages. J. Wiley and Sons.

CHAPTER XII

FURTHER ASPECTS OF THE FUNCTIONAL SPECIFICATION OF PROGRAMMED FUNCTIONS

This chapter expands aspects of the functional specification in the area of programmed functions.

1. INTERACTION HIERARCHY

A "session" is a series of one or more commands issued by the user. The user must implicitly or explicitly specify a username before he can run a session.

A "command" is the normal unit of interaction between user and the computer-aided wagon design system. A command is split into one or more tasks.

The task (also referred to as an operation) is the unit of work which the user considers to be processed indivisibly by the system. It is thus the lowest level at which he can express his requirements for system resources. Each task carries out a logically self-contained operation. It is axiomatic that a task is always an atomic operation as far as the permanent storage of information is concerned. Any change in the state of the database must take place instantaneously as far as it is concerned so that the self-consistency can be preserved. It is possible to interconnect tasks by passing messages between them while they are executing. Such messages may not be visible at the command level.

An executable collection of tasks and/or commands and/or invokable programs is called a procedure.

2. COMMAND TASKS

Programmed functions in the system can be regarded as falling into four main categories: user environment tailoring, utility, design, and data management. Data management functions are introduced in the following chapter.

(1) Utilities

(a) Wagon Designer. These functions are a subset of what is usually available in an operating system command language, but for the wagon designer it is desirable that they can be invoked from within the computer-aided wagon design system and have conformity with the system's concepts and constructs.

The first group of utility functions deal with text file management. Text files can be used for data preparation for design tasks, exchange of data with other systems, and formatting and editing output from the system for use in documents. File directories are hierarchical catalogues of program and data files containing their definition and logical instance pointers. Commands required include: copy files, create files, delete files, display or list contents of files, rename files, screen mode editing of files, delete old versions of files, set file characteristics (such availability, history and security), display contents of directories, display file characteristics, append files to another file, and compare files to find

differences. Usability features such as flexible file selection, confirmation of request and logging of results should be available.

Definition, monitoring and removal of batch or background tasks is another category of utility functions that should be provided. Usability features should include definition of scheduling, priority, and resource limits. Print and plot to obtain output are special cases of background jobs in which specification of number of copies, format and such like may be required.

Other technical office functions such as engineering document processing, spread sheeting and electronic mail will also be required.

(b) System Developer. System development and management facilities such as resource allocation, usage logging and program compilation are assumed to be available from the operating system of the computer in an appropriate form.

(2) Design Tasks

A design task applies a solution method to a model with given input values to obtain new values for the dependent design entities. Because design models and methods are developed by specialists, the author notes their existence (chapters II to VIII) and derives the requirements to facilitate their inclusion into the system and facilitate their use by wagon designers, but has not attempted development of the models and methods themselves. A range of sophistication of design tasks, for example, from simple spring stressing calculations to complex vehicle dynamics analysis should be implemented to support the iterative and continuous refinement nature of design. The benefits of rationalising and standardising should be balanced against flexibility and ease of

incorporating new techniques and products.

(a) Scope of a Design Task. At varying levels of detail, the data elements used by one operation may themselves be the output data from another operation. Each instance of a data element may be used in a number of operations. The interconnection between the operations or the relationships between the data elements cannot always be determined beforehand (as when the decision made by the computer depends on more than one condition or when the decision is a subjective one made by a human expert). The level of detail for these relationships or the granularity of the models depends on a number of factors. The predominant issue is the lowest level of programmed design function required by the wagon designer. At what stage would the wagon designer like to interrupt the process, store values, perhaps to compare values and obtain feedback on the request or to continue some time later? Other modifying factors include in-core storage limits, in-core data versus database accessing speeds, and the availability of source code for the application programs.

(b) Stored Design Strategies. Collections of one or more design tasks may be stored for the wagon designer to execute if he so wishes. These design strategies will enable the system to choose and execute the appropriate methods, to choose models, and to use or update data according to the model's requirements in the process of seeking the user's specified goal. At the simplest level default methods for solution of the specified model and dependent design entities should be determined by the system. Choice in methods and models may be restricted by design management for certain users. Facilities will be required for module to module communication and synchronization of

module execution.

(c) Execution Monitoring. The facility to monitor the execution of these design strategies should be provided for the wagon designer. After initiating execution the wagon designer may wish to view the changes in value of specified design entities and identify the design tasks performed or he may just want to know that execution started normally and to be notified when execution is completed. At the completion of any design task, any data manipulation task could be performed on the database.

(d) Control Over Execution. The requirement that the wagon designer should be able to change the flow of processing during execution of a command implies that he can suspend a command or quit a command, change the value of parameters, initiate another command or restart the suspended command. If the intermediate results supplied from each design task are saved in the database, then if the computer system fails the execution could be restarted from an intermediate stage close to the point of failure. Being able to stop the execution of a command after the data is saved gives the user the option of either continuing the command immediately or stopping to continue later. He can also go back and restart the process from any desired step. Conditional break points or stop locations between one design task and another with actions to be performed on breaking execution, could be specified by the wagon designer. From these break points the wagon designer could opt to continue one task at a time or continue until some condition is satisfied. On detection of an error when reading from a file, the system should offer the opportunity to enter any command, return to the input file or end the file input. Whenever the system

requires information, execution should also be able to be suspended or quit.

3. MODIFICATION AND EXPANSION

Although the boundary between system developer and wagon designer will shift and blur somewhat it is still possible to distinguish them fundamentally and, therefore, to tailor the computing tools supplied to each.

(1) System Developer

Enhancement and addition of complex models is the realm of the system developer for reasons of efficient use of computing resources, quality assurance, and skill in implementing computing systems. In addition to the utility functions described under section 2(1), facilities will be required to assist in the addition and modification of design, utility, and other types of programmed tasks to the system. The use of decision tables is one method of high level specification of programming tasks.

(2) Wagon Designer Programming

A programming facility for wagon designers that is suitable for both immediate execution and non-interactive execution will allow automation of the routine, tailoring to individual needs and addition of knowledge to the system. Such a facility would allow users to write procedures involving all commands and functions provided by the system and to use additional features.

One means of creating such procedures is to record, in a form that can be edited, the user's operations while using the system. Creation through text file editing is another means.

The objects of a high level programming facility should include numerical, boolean, and literal scalar variables and constants. Accuracy of numeric scalars should be at least that achievable with 32 bits and 64 bit precision should also be available. Dynamic arrays of one or more dimensions of scalars should also be available. Database entities are also objects in the programming facility. Manipulation of symbolic models, where the solution is in terms of variable names is another desirable feature.

Operations should include: common arithmetic operations and trigonometric, exponential, and geometric functions; relational and logical operators; string operation; array (and matrix) operations; definite integration; equation and equation system solving; discontinuous functions such as sign, maximum, and minimum; empirical or user defined functions; and partial derivatives.

Programming statements would include assignment, choice and repetition control structures, user interaction, scope definition of variables, procedure definition and naming, invocation of other procedures and passing of parameters, and the connection and use of text files. The ability to link procedures with routines written in other encoding languages is also required. Error handling features would also be desirable.

The system should also allow for the formulation of any question regarding a selected model by specifying any variables as the dependent variables. Free manipulation of the input and output variable subsets

should be permitted. It should be possible to automatically check feasibility of a design given the parameters, or to proceed from the feasible boundary to the design parameters. In the mathematical optimisation type of model, the wagon designer should be able to specify the objective function(s) and constraint functions as well as be able to use inbuilt procedures.

Ideally the wagon designer should be able to interactively enter arbitrary expressions and relationships between design entities (the models) and interactively define the design entities and solution methods.

CHAPTER XIII

FURTHER ASPECTS OF THE FUNCTIONAL SPECIFICATION OF DATA MANAGEMENT

This chapter expands aspects of the functional specification in the area of data management. This chapter covers requirements; implementation details for some data structures and data elements are presented in chapter XVII.

1. THE SCOPE AND CHARACTERISTICS OF A DESIGN DATABASE

The database is the repository for data used by the wagon design system. Information on a design can be classified into a number of groups.

Geometric information comprises the definition (parameters, shapes and location) of the three-dimensional (3D) geometric elements in a part and their topology and connectivity (or in other words, the relationships between geometric elements).

Technological information consists of form, position, direction and dimension tolerances; material characteristics; surface finish, surface treatment and heat treatment definition and association to geometric elements; and identification of common part features and their associated properties.

Documentation information includes, drawing detail to 3D geometrical model relationships; dimension to geometry relationships; and drawing note and specification definition, and their relationships to geometry.

Assembly information includes part-instance location in assemblies; mating features of parts in assemblies such as clearances and type of spatial relationship (planar, cylindrical, etc.) and their relationships with geometry; the relationship between connected parts (for example, rigid attachment, relative motion conditional on force being beyond threshold value and translational and rotational constraints on relative motion).

Administration information consists of creation, and release and change control information on parts and assemblies.

Functional analysis information comprises mass and section properties, force and motion equations governing suspension connections, elastic or inelastic deformed shapes, stresses, loads, accelerations, displacements, and so on. Analysis information is in general related to geometry, material properties, and assembly information.

The scope of design data will make a database system designed solely for graphics unsuitable for an integrated scheme.

Data may be stored not only for wagons and their components, but also for raw materials, permanent way and trackside structures, ladings and loading/unloading systems.

In general, design databases can be characterised by many different data types and many relationships between these data types. Each data type may only have one or two instances or it may have many thousands (for example, finite element displacements). While part of an active project, the design description is under constant change, but, when it describes an approved design, there are fewer changes and on older designs slower access times may be tolerated. There is also information

which is referenced by wagon designers but cannot be changed at all by them. Everything from simple queries to complex analysis codes run against the database; queries can span long time duration and involve a large number of records.

2. DATABASE ADMINISTRATOR

The system developer influences the database in two ways. Firstly he creates the application programs that use the database. Secondly there is a class of system developer that can be distinguished from other system developers in that this class of user controls the database system, that is, he is the database administrator. Database administrator represents a function which may in reality be one of many responsibilities of one person or alternatively this function may be spread across many people.

The database administrator is responsible for what information will be stored in the database, how the data will be stored and accessed, documentation of the data and relationships stored in the database, what authorisation checks and validation procedures will be performed, what backup and recovery procedures will be used and how the system should be tuned to meet changing performance goals. Further details on the functions of a database administrator are available in Date (1981) and Atre (1980).

3. FEATURES OF A DATABASE

The features and qualities of databases were introduced in chapter

IX. The main points are repeated here and some aspects expanded.

The textbooks, Martin (1977) and Date (1981), have been used for guidance in this area as they have interpreted the reports of bodies such as CODASYL systems committee, SHARE and GUIDE, ACM, and ANSI X3 SPARC/DBMS, and other individuals who have studied database issues at great length.

(1) Data Definition and Manipulation

The basic functions of data manipulation are: retrieve and display data, delete values, insert values, update or edit existing data. In addition facilities to output data to a printer/plotter or to an operating system text file and to input data from a text file are required. Sorting, formatting, grouping, selection and statistical calculation facilities should be available for data retrieval.

Data should be addressable on a straight forward and systematic basis. Users should not need to specify how or where it is stored but rather they should be concerned with the data's logical structure. Data should be addressable at the data element level and selection should be possible through specification of multiple keys.

Similar facilities that define, delete, edit and display data structures and their relationships are also required.

All functions should be available interactively or through an application program.

(2) Wagon Designer Query Language

A powerful and simple interaction facility should be available which permits users to search and update data, and to generate reports

or documents without application programs being written. With such a facility unanticipated and spontaneous requests for data can be handled quickly. It is desirable that such a query facility be invokable from within the computer-aided wagon design system and have conformity with the system's concepts and constructs.

(3) High Level Programming

A programming interface that permits simple and powerful data requests should be available. This interface should insulate the programmer from the complexities of physical storage layout and addressing.

(4) Data Validation

It should be possible to define validation checks on data to be performed when changes are made. These procedures should check the data against acceptable ranges, check the data contains legal characters, and check data values for consistency with other data values. Referential and key constraints should be implemented.

(5) Accuracy

Accuracy implies that all the data types available in the encoding language and in the user's programming language should be available in the database management system. This is particularly important to avoid truncation of numbers.

(6) Reconstruction of Data

Two common ways to applying changes to data are:

- a new physical instance is produced by the application of zero or more changes to the current version;

- changes apply directly to the current instance.

Integrity requires that the user's actions are monitored or logged and periodic copies of the database are made in order to restore the database. Monitoring will also help to determine the source of failure or security breach. The number of successive instances of the data that are kept and the number of days for which it is possible to reconstruct instances are two aspects of integrity management. The date and time of data creation, of the last update and the date and time that the last backup copy was taken will have to be maintained. Facilities should be provided to enable recovery from database corruption including that caused by system failure and abnormal program termination.

(7) Performance Tuning

Wagon design does not have a well understood and specific set of operations. The database administrator will not know exactly how the database will be used or how often. It will be necessary to adjust or even fundamentally change the organisation of the database after the system has come into use and the usage pattern has been perceived. Usage will continually change as applications evolve and users gain experience.

Frequently referenced data may be stored on disk and the data that is rarely accessed can be stored more cheaply on magnetic tape if response time is not critical. Archiving (medium to medium copying) for long term storage should be possible. Maintenance of the date and time of the last access to data will aid archiving. A command will have to

suspend execution if data is not immediately available.

Effective performance improvement has two requirements:

- performance statistics monitoring and collection, and design aids must be available to the database administrator;

- physical data independence is necessary in order to change the physical storage system without affecting the application programs.

(8) Data Independence

Data independence is important for achieving a stable database which facilitates change.

The wagon design system does not at present have a large commitment to computing, as evidenced by the small investment in existing computer-based programs and data, and yet the need for future database systems to be compatible with existing packages will be a major constraint.

Earlier application programs and even many of today's programs are data dependent. The way in which the data is structured and accessed in secondary storage is heavily influenced by the application program and the knowledge of this structuring and accessing is built into the application program itself. For example, a random access file type may be chosen and then the means of addressing records is coded into the program, the precise form of which depends upon the input/output software. This application is data dependent because it is impossible to change the storage structure or access strategy (for example, to indexed sequential) without affecting the application program. An important observation is that the portions of the program requiring alteration are those that communicate with the file-handling software. These portions are quite irrelevant to the problem the program was

written to solve.

Therefore the provision of a high degree of data independence is of major importance in database systems. Date (1981, p13) defined data independence as "the immunity of applications to change in storage structure and access strategy". Martin (1977, p30) identified two types of data independence (refer fig. 36): "Logical data independence means that the overall logical structure of the data may be changed without changing the application programs." (Except when data that the program uses is removed). "Physical data independence means that the physical layout and organisation of the data may be changed without changing either the overall logical structure of the data or the application programs."

To achieve logical data independence any single application program's view or end user's view of the data needs to be severed from the global logical data representation. It must be possible to add new fields or new relationships between data elements without having to rewrite the programs which use that data. When changes are made to the physical representation of the data or hardware, these changes should be reacted to in the database management software and should leave the application programs untouched. If the system imposes a rigid structure on the database, changing user needs and changing technology offering improved efficiencies will result in unnecessary effort spent on modification of existing software, rather than progress on new applications.

	No change in (other) application programs	No change in the global logical data description	No change in the physical storage organisation
- A new application program is added, using existing types of data	X	X	X
- An application program uses a changed representation of existing data (e.g., floating point instead of fixed point)	X	X	X
- New record occurrences are inserted, or old ones deleted	X	X	X
- The physical organisation of data is improved; possibly different representations are used	X	X	
- The addressing methods are changed	X	X	
- The data is moved to a different type of volume	X	X	
- The software is changed	X	X	
- The hardware is changed	X	X	
- A new application program is added, using new types of data	X		
- The global logical data description is improved, or new relationships between data types are created	X		
- Two databases are merged	X		

Figure 36. Data Independence Capabilities
(Source: Martin, 1977)

Some of the aspects of the physical database that can be changed are:

- A character string may be stored many different ways and in any of several character codes, for example, EBCDIC, ASCII.

- A number may be stored as a character string or in some internal form. The base (for example, decimal or binary), scale (fixed or floating point), mode (real or complex), and the number of digits must be chosen.

- The physical data item may not exist; instead its values may be calculated from several stored data items. For example, total weight may be calculated by summing a number of weights. Total weight is an example of a virtual field. It is not possible to directly create or modify an occurrence of a virtual field.

- A stored file may be contained within one storage volume or it may be spread across several volumes of differing types and sizes. The address of the storage devices may be changed. The file may or may not be accessible via indexes, hash-addressing or other means. The stored records may or may not be blocked. The records and blocks may be of fixed or variable length.

None of these variations should affect the application programs except in performance.

The logical structure of the data may be changed in some of the following ways:

- The units in a numeric field may change, for example, from inches to metres.

- Data may be represented in storage by coded values. For example,

an application may require a character string for part name (handbrake lever, draftlug, etc.) but the part name may be stored as a numeric field with 15 = "Draftlug", 36 = "Handbrake lever", and so on.

- Two or more existing records may be combined into one, or an existing record may be split into two or more records. For example, a part record may be split into part and drawing records. This would mean an application program's logical record would consist of some subset of the stored record or that the logical record would contain fields from several stored records.

- A new field could be added to an existing record.

- Existing fields and records could have their names changed.

- Relationships between records could change type or name.

- The method of maintaining privacy is changed.

If the computer-aided wagon design system database were to be integrated with other databases, data independence will aid the evolution to distributed database operation. Data independence would make it possible to change the physical location of data without affecting applications.

To maximise the usefulness of an on-line database, the data must be structured in a clear well-though-out manner that is independent of the needs of any single application.

(9) Simplicity in the Conceptual View

The means of representing the conceptual view should be simple and neat, one with which the most computer-naive user to the most computer sophisticated one can interact but not prohibiting specialised superimposed user views. Such a conceptual schema would simplify the

potentially formidable task of the database administrator.

"In many systems pointers are used in the logical representation to show relationships between data items... A problem with logical pointers is that as more and more relationships are added between data items the overall collection of pointers becomes highly complex and it is difficult to represent the overall logical view of the database with clarity. There are pointers to pointers to pointers. In some cases... multiple pointer links in the user's representation of data can be highly misleading. There is no need for the complexity that exists in some logical data structures." (Martin, 1977, p43).

Date (1981) interpreted simplicity as ease of understanding and ease of manipulation of the conceptual view. He pointed out that this does not imply that it should be minimal in any sense, rather that it is simpler (more usable) from the user's point of view.

(10) Data Dictionary

A comprehensive data dictionary facility is required. It should contain information on data items and the external, conceptual (and logical) and internal schemas. Its information should be available for use by both humans and the system. The description of data structures and so forth should have a status attribute and they should be subject to similar access controls to the actual data. Backup, recovery, tuning, and other management utilities should be available. As a database in its own right it is subject to the same requirements as the design description database.

The wagon designer should be able to define entities and

relationships in the data dictionary.

4. ADDITIONAL FEATURES OF A DESIGN DATABASE

Other features of computer-aided wagon design database management are presented in this section.

(1) Multiple Views

The database acts as a communication link between wagon designers and computerised design tasks; it is accessible and updatable by various programs and users. Application programs will use different representations of the same design entities to achieve efficient solution to the particular problem they are designed to solve. The views may differ in units for attributes or in the level of detail of information describing a design entity (particularly noticeable with the evolutionary nature of design). Definition of external views will provide the necessary mappings, insulate users and programs from conceptual schema change, simplify the user's or programmer's view of the database, and provide automatic security control.

(2) Documentation of Design History

A record must be made of at least the important parts of the design path. This history of the actions taken and the decisions made is important for the understanding of the wagon design (which is a prerequisite for modification) and also that the insufficiencies of the system can be detected and corrected (that is, performance monitoring and tuning). A summarised history should include a brief description of

the data at a particular time, who or what changed it, why it was changed, and a brief description of the new data together with its creation time. A fuller description of the data is required where backtracking to a previously discarded alternative is a possibility. This history of changes may also provide the means to reconstruct the database for integrity purposes.

(3) Multiple Alternatives

During the process of design, alternatives, each with some degree of difference in their attributes, are often generated. These alternatives reflect the tentative nature of design. If the system supports sets of possible solutions then the wagon designer can investigate one alternative, come back to another, form yet another from these two, and so on. The database management system should provide a capability for storage and management of multiple alternatives of a design. It should also ease the development of alternatives. It should be possible for copies to be taken of all or part of the design description for use in another alternative design.

(4) Data Structures

The conceptual schema should act as a stable and enduring foundation for the system. It should be able to represent the associations inherent in the data and not force data into structures which do not represent its inherent nature.

"The conceptual schema should not have to be changed unless some adjustment in the real world requires some definition to be adjusted too, so that it may continue to reflect reality." (Date, 1981, p.474).

For example, it is frequently necessary to expand the conceptual schema to reflect a larger portion of reality.

The requirement for handling complex objects is derived from the wagon designers' need to refer to objects which can consist of tens or even hundreds of database records, for example, arrays, entities composed of a number of relations, and relations which possess class attributes as well as their own attributes. Accessing this data on a record basis would involve cumbersome and tedious accesses to several instances of many different record types. Users need to be able to refer to these complex objects as one entity.

(5) Hierarchies of Design Tasks

The design process can be thought of in a top-down fashion in which complex design problems are recursively divided into a hierarchy of subproblems whose solutions are conceptually simpler and can be assembled to provide a solution to the original problem. Design tasks were introduced in the previous chapter as a way of modelling subproblems in a manner that did not constrain the design process. A design task was defined as the application of a solution method to a model to obtain values for the dependent entities given the input entities. The results of a design task were stored in order to support execution control and from the database they could be inspected by the wagon designer and/or passed to another design task.

The database should also contain the possible sources and uses for a design entity (possibly in the data dictionary) so that this semantic information is available to the wagon designer. Database integrity and meaningfulness will be enhanced if how and when individual instances of

design entities were created is stored, and if how instances have been used in other design tasks is also stored. This implies maintaining the relationship between dependent and independent variables, solution method, and model for each design task. For example, the volume and density of a part may be used to determine its weight and in an alternative design of this part its volume and weight may be used to calculate another instance of density. There may be a number of alternative methods for solving the equation for these variables, including that of inputting an estimate from the wagon designer.

For each design task the constraints necessary can be defined. For example, if a stiffness matrix and a load matrix are used to calculate a displacement matrix all that is necessary is that the number of rows in the stiffness matrix and in the load matrix is the same and that the stiffness matrix is positive definite (that is, the matrix equation is solvable). Whether the load matrix was derived using the same stiffness matrix (if there were prescribed displacements) is irrelevant to this design task. Only for a larger model (for example, the calculation of displacements given the structure and load definitions) does this become important, in which case this constraint would be incorporated in the model/method specification.

This approach can also be used to store standard results or capacities. For example, a pin diameter and the ANZR design model and method could be stored, together with the pin's load capacity calculated according to this model and method.

The stored design task relationships would serve as a basis amongst the wagon designers and application programs. Relationships between design entities and between design entities and wagon designers can be

found by searching through the database.

(6) Concurrency Control and Update Propagation

Concurrent access can introduce integrity problems. If a design task reads data off the database while another design task is updating the data there may be integrity violations. A wagon designer may work for hours on a set of data in an interactive mode. These long conversational transactions cannot be handled adequately by the conventional locking or lock-free concurrency control mechanisms. These mechanisms work well when the transactions are short in duration, of the order of seconds. Additional integrity problems can occur when one design task uses data created by an earlier design task. If this earlier design task is reiterated or if any of the data used by this earlier design task is changed, then the current design task's results may be invalidated. Each design application, because it derives data from data, adds to the data dependencies and consistency problems.

Most schemes proposed to counter these problems basically involve taking temporary copies of the consistent database for the case studies, controlling the use made of the original data and applying update propagation or flagging to the total database only when the modified data is re-integrated with the global database (Buchmann, 1984; Rasdorf and Kutay, 1982; Eastman, 1981; Bandurski and Jefferson, 1975; Tsubaki, 1975). These schemes assume that changes are made directly to the one current instance of the data as opposed to an application of changes producing a new physical instance (that is, version). Buchmann (1984) claimed versioning was costly, and difficult to avoid redundancy because the alternatives are often extensively different and most are eventually

discarded.

Applying changes to the database in place however does also have its problems. Design tasks may be regarded as transforming the database from one state to another. Consistent processing by a design task is dependent on the successful execution of all the design tasks which compute data for the current design task. Eastman (1981) and others regard the database states succeeding an iterated design task as invalid requiring execution of their respective design tasks again. However calculated or estimated data cannot be described as true or false, valid or invalid. They can however be described in how accurately they model the real world given the start state. What is true or false is whether a particular database state is related to another database state. The alternative solution generated when a design task is iterated may or may not be the currently favoured one; it may or may not be proposed as the solution by the wagon designers responsible for that particular area of the wagon design. If this new alternative is not approved then the original solution applies and the succeeding transactions are not invalid. Therefore to describe an iterated design task as invalidating dependent transactions is too simplistic.

Conventional concurrency controls can be applied to design tasks when viewed as transactions modifying the database in place.

Propagation of changes throughout the database all-at-once may lead to extensive updating and high overheads for any database update as transactions can be repeated along many different paths. Such automatic update propagation is impractical. It would be expensive in its use of computing resources and may involve much needless testing and conflict resolution. Many transactions will have to be maintained by user

actions. It is no problem to infer, for example, gauge pressure from absolute and barometric pressures. However, the data resulting from decisions involving such intangibles as customer-preference or designer's judgement can hardly be inferred automatically. Wagon designers also estimate likely values on occasions (rather than analytically determine a value) so that the design process can proceed. Later such values may be checked by computing values in a full analysis. Analyses themselves may ask the user to supply missing data. If such data is entered into the database then the fact that a wagon designer created it needs to be recorded. When to check integrity is often best controlled by the designer. It is necessary to know who has used the data subject to change so that they may be notified.

A design database should support the attachment of messages to particular data items so that others may learn of special assumptions or considerations.

An important consideration in the deletion of discarded design data is the maintenance of referential integrity. In the database there should not be references to non-existent data. In particular data that was used to generate more data cannot be changed or deleted unless this integrity constraint is met.

Because a history of the design process should be maintained as should design alternatives, versioning would seem the better approach. This means that old versions must be kept but they can be archived relatively cheaply. For long term storage only a description of the revision may be retained. To save storage space, referencing existing data items should be done and only modified data should create a new version. Data structures to record the relationships between the input

state and output state of the data base can be easily added for each new programmed design task.

Access controls can automatically provide a status check. Release of information to other wagon designers is essentially an issue of access rights on a global integrated database. The checking and approval procedures determine whether the wagon designers are confident enough to share one or more alternatives with other designers (and therefore have the updates propagated). The up-to-date status of the design is just a particular view of the data and is not necessarily a distinct physical copy. Implementation of different access controls on different versions ideally requires the ability to control access to sets of tuples in a relation, but could also be implemented with multiple instances of the same relation (that is, private databanks from which data could be loaded into the same data structures in a global databank).

This status information can be used to manage the workflow and reports could be obtained on percentage complete, percentage in review, where versions are used, when they were changed and by whom, who checked the change, who approved it, and what changes were made.

Concurrency controls need only be applied to the creation of a new instance (not as in the case of in-place updating to the only copy of the data while it is being modified). This allows practical transaction management that could also have an effective date as well as creation date (that is, the date on which it is to become the new revision).

By always creating a new instance of modified data which is date and time stamped, access to the latest approved version can be guaranteed. And by the same token the database can be queried to find

what data was used to create any particular results.

Checking consistency of the data used by different application programs can become just another design task where such programmed checking is possible. (For example, dynamic loads calculated from a dynamics program can be checked for consistency with those used to calculate dynamic stresses which in turn may be checked against those used in fatigue calculation.) These checking runs can be delayed until the wagon designer requires their execution. Checking consistency is important where there is a choice of several means of generating the required data.

5. REFERENCES

ATRE, S. (1980) Database: structured techniques for design, performance, and management. New York, John Wiley and Sons. 442p.

BANDURSKI, A.E. and JEFFERSON, D.K. (1975) Enhancements to the DBTG Model for Computer-aided Ship Design. Proc. ACM Workshop on Databases for Interactive Design, Waterloo, Ontario, 1975. p.17-24.

BUCHMANN, A.P. (1984) Current Trends in CAD Databases. Computer-aided Design, 16 (3):123-126.

DATE, C.J. (1981) An Introduction to Database Systems. 3d.ed. Reading, Massachusetts, Addison-Wesley. 574p.

EASTMAN, C.M. (1981) Database Facilities for Engineering Design. Proc. of IEEE, 69(10):1249-1263.

MARTIN, J. (1977) Computer Database Organisation. 2d.ed. Englewood

Cliffs, New Jersey, Prentice Hall. 713p.

RASDORF, W.J. and KUTAY, A.R. (1982) Maintenance of Integrity During Concurrent Access in a Building Design Database. Computer-aided Design, 14(4):201-207.

TSUBAKI, M. (1975) DPLS:database, dynamic program control and open-ended POL support. Proc. ACM Workshop on Databases for Interactive Design, Waterloo, Ontario, 1975. p146-160.

CHAPTER XIV

COMPARISON OF EXISTING SYSTEMS

In the first part of this thesis the NZR wagon design system was described and the results of a literature search for wagon design computer aids presented. The second part dealt with the general functional specification of a computer aided wagon design system that would best assist in the attainment of NZR objectives.

This chapter reviews some of the existing solutions to the general problem of integrated computer aided design. Subsequent chapters describe the implementation of a prototype, system and discuss the prototype, further work and implementation planning.

1. INTRODUCTION

Developing a software system that presents the illusion of a single, integrated, consistent database and set of applications in a manner convenient to use has become an objective of many CAD researchers. While there is only a limited amount of research conducted in the field of computer aided wagon design, efforts to develop integrated design systems are under way in many other engineering fields. A review of selected efforts to provide integrated systems is valuable both in terms of solutions adopted to problems similar to those found in integrated computer aided wagon design and also to determine whether their capabilities would make them candidates for revision to the wagon design environment.

2. REVIEW OF SELECTED INTEGRATED SYSTEMS

(1) Integrated Civil Engineering System (ICES)

Leroy and Green (1981) provided an overview of ICES, the main points of which are summarised below.

ICES was developed to correct deficiencies in the engineering computer programs and programming languages available at the time (many of them still evident in application programs and languages in use today). These deficiencies were:

(i) Rigid input formats and familiarity with computing required before an engineer could effectively use the application program.

(ii) Linear nature of the solution process utilizing limited and fixed data structures which allowed only limited user control over program sequence, and restricted problem size.

(iii) Output results in a fixed form, or complex, inflexible and "unnatural" (that is, not in an engineering discipline oriented language) specification of requirements.

(iv) The large percentage of system specific programming in the more comprehensive systems.

(v) Lack of a convenient environment for engineering programmers in which to expand and enhance capabilities.

The main thrust of the ICES research and development effort was undertaken by the Civil Engineering Systems Laboratory at the Massachusetts Institute of Technology during the years 1964-1972 at a cost of approximately US$4-$5 million and was sponsored by various companies and public bodies. Today ICES development is carried out by

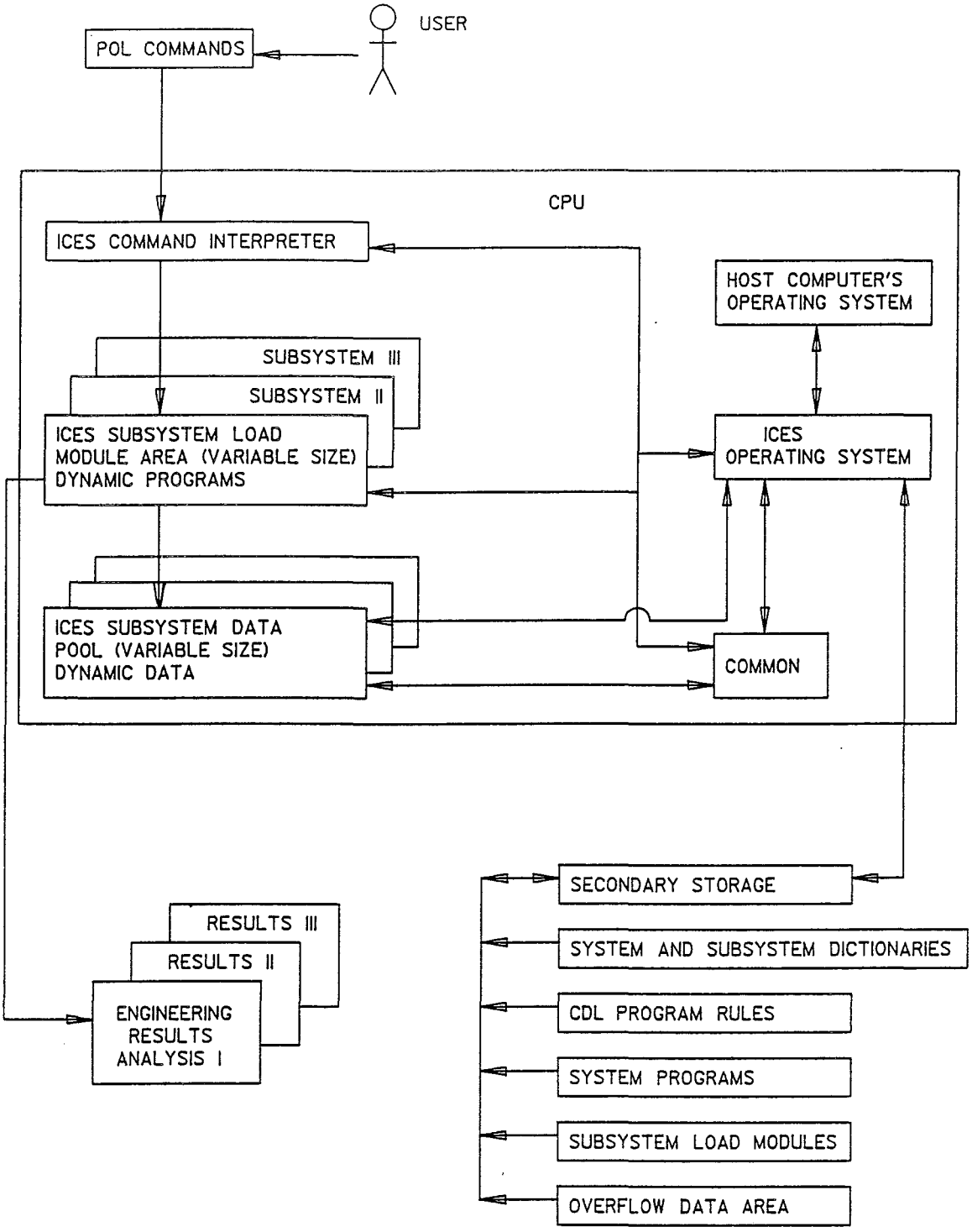


Figure 37. ICES Architecture
(Source: Leroy and Green, 1981)

the users. ICES operates on a number of large computers of various manufacture including CDC, IBM, UNIVAC, PHILLIPS, SIEMENS and DEC.

ICES was essentially made up of two components (refer fig. 37):

(i) Basic system programs;

(ii) Application subsystems.

The application subsystems ran under control of the basic system programs which in turn ran under control of the host computer.

The user interface was through problem oriented language (POL) commands which were discipline oriented and made of words from the English language. The user was free to employ these instructions in whatever order was sensible to solve the problem and were conveniently accessible in interactive or batch or combined operation. ICES allowed an analysis to be interrupted at specific points and restarted later. In the basic system, there were a set of commands by which a user stored data, directed processing, and printed and/or displayed results, otherwise the commands and responses from the computer were in the subsystem language. Subsystems were collections of individual programs each designed to perform a specific task, while the whole subsystem was designed to solve problems in a particular field.

For the programmer, ICES provided convenient and common facilities to interpret POL commands, execute the appropriate modules of the program, manage storage requirements of jobs at the time of running, and to set up sub-system POL's. For the user, ICES provided a common set of language conventions enabling experience to be transferred between subsystems and the capability to transfer data from one subsystem to another.

Some of the ICES subsystems were:

ETHIC Allows a user to write, try out and store "codes" to be used with code checking and item selection facilities of other programs.

SIFTER General purpose report generating Program.

SECDES Enables the user to investigate the load capacity of a structural member composed of arbitrary materials.

SPEC Allows text to be stored in paragraphs which may then be edited and printed according to a desired format.

STATS Provides facilities for manipulation of columns of data including statistical functions.

GRAPHIC Displays data in the form XY plots or three dimensional objects.

STRUDL Calculation of stresses under given loads and dynamic behaviour of framed structures and continuous materials.

TABLE Allows users to create and manipulate tabular data.

Two languages (ICETRAN-ICES FORTRAN and CDL-Command Definition Language) were provided for the development of subsystems utilizing the ICES capabilities. Other capabilities provided to run the subsystems were:

(i) Command Interpreter to interpret input and supervise execution.

(ii) Dynamic program structures to dynamically link the programs used in processing of commands.

(iii) Dynamic memory allocation to dynamically organise data and programs in primary memory based on processing requirements of the moment.

(iv) Management of data on secondary storage.

ICETRAN was an extended FORTRAN with expanded operations and capabilities to utilise dynamic data structures, dynamic program structures and data management. These facilities were implemented through FORTRAN-like statements and not the subroutine CALL statement. A pre-compiler translated these statements to subroutine calls resulting in conventional FORTRAN.

ICETRAN statements existed to load a program into primary memory if not already there. The programmer did not need to know the history of the process and programs no longer needed to be removed from primary memory. At the time the total size of each ICES subsystem was many times the size of primary memory yet only a small subset was required at any one time. Communication between modules was provided by means of a special common block. Dynamic program structure enabled a programmer (using the same linkage statement) to switch to a different set of programs in, for example, an iterative process where the alternative program names were provided at execution time rather than at compilation time.

Dynamic data treatment allowed the programmer to use data logically without significant concern about size or physical location of data. The size and structure could be specified at execution time. The system programs shifted arrays or portions of arrays around in primary memory or between primary and secondary storage; moving out arrays not currently needed and moving in when referenced. With this system the

size of problem that could be tackled was limited by capacity of secondary storage rather than capacity of primary memory.

Typing of data was also reduced so that double precision and real numbers could be mixed in the same array. Each row of an array could have a different number of elements. Arrays could be partitioned into subarrays that varied in size and structure. These subarray structures could be operated upon in the same manner as total arrays.

The internal data structure was so implemented as to be easily developed to take advantage of new and efficient storage and retrieval techniques as they became available.

Data management facilities made use of direct access secondary storage. ICES files were contained within operating system files but still had full file management functionality, for example, deletion, protection definition, appending and renaming. The logical records in an ICES file could be updated, extended, truncated or deleted.

The CDL enabled the programmer to specify structure and processing of the POL commands. The output of CDL processing was a dictionary and a set of rules.

Gaggero (1981) reported on a revised and expanded CDL called WESPOL that provided a tool for the implementation of a set of POL commands. The WESPOL compiler processed the definition of the POL commands specific to an application and presented them to the WESPOL Command Interpreter (CI) in an application independent form. The CI performed input functions, identified user commands, analysed their elements and invoked execution of the appropriate application modules. Both the compiler and interpreter were written in portable FORTRAN. WESPOL was developed to reduce programming complexity, add flexibility and promote standardisation.

ICES was developed partly because of the shortcomings in available computer operating systems of the time in the context of large scale engineering processing. The implementation required extensive programming in assembly language (due to limitations in FORTRAN) to supplement the existing operating systems provided by the computer manufacturer in areas such as input/output to temporary storage files and memory allocation. Use of assembly language has reduced ICES portability.

These limitation have now largely been removed in modern programming languages and operating systems (Eastman, 1976), for example, ADA, MODULA-2 (an extended PASCAL featuring modules and concurrent processes), C, Unix and VAX VMS. Use of these languages and operating systems makes for more cost efficient development and maintenance.

C, for example, as well as having high level features such as data structures, also has facilities that possess some degree of portability such as to spawn a subprocess or task; to either wait for that subprocess to finish or to continue immediately with the parent process; to communicate data between processes; to provide accounting times; to wait a specified length of time; to handle error or user interrupts; to run an operating system command; change process priority; and to execute a named file. This last feature enables a process to elect to change its program (that is, the instructions that it obeys) as it runs (Banahan and Rutter, 1983).

VAX VMS, an example of a modern operating system, has a 4GByte virtual address capacity (a very large virtual memory), a linker that allows sharable images to be subsequently linked with other modules, and

a librarian to facilitate updating and management of libraries of routines (both object modules and sharable images). Library facilities allow separate compilation of modules and export/import of parameters between modules. This enables linking to proprietary code. Balling and others (1986) have developed an interface for multitasking computers with which the optimisation program Optdes.BYU could be linked to any stand-alone software without the need for new interface code.

Eastman (1976) described ICES as a first stage system which provided an enhanced machine environment for loading, controlling, and executing analysis programs, and for input/output to secondary storage and an enhanced user environment. This level of system reduced calculation costs by allowing more sophisticated and sensitive analyses than previously possible. Data preparation and reporting was still a major issue.

Iterative use of an application was possible by editing data between iterations. There were limited file reformat capabilities to reorganise and edit data for another application (an exception was the STATS subsystem). Mostly the user could only access data with the subsystem that put it there. To input data to a sub-system from another required recourse to operating system text editors after obtaining a suitable sequential dataset from either input/output files or by converting the subsystem dataset. Ad hoc reporting could not be achieved directly.

ICES is claimed to be the most widely used large scale engineering software in the world although there are others. For example, the GENeral Engineering SYStem (GENESYS) is a library of computer programs which include : Structural Analysis, Roadway Design, Geotechnical

Analysis, Building Design and Detailing, Construction Management and Fluid Distribution. All data is prepared in a standard manner using a POL. The engineer can control the sequence of calculation and the output from one subsystem can be used as input for another subsystem. A programming language called GENTRAN, which is a superset of FORTRAN, is provided.

(2) Database, Dynamic Program Control and Open-Ended POL Support (DPLS)

The development of DPLS (Tsubaki, 1975) was initiated in 1971 with the primary objective of supporting all the information system needs of the Chiyoda Chemical Engineering and Construction Company, a large process plant design and build firm. DPLS ran on an IBM 370.

It had three major functions : total engineering and business database management, dynamic program control, and open-ended POL. It was aimed to provide a user oriented extension to the data management, job/task management, and compiler languages available in the computer operating system.

Criticisms levelled at DBMS's of the time included :

(i) No semantic or conceptual view of data and unsatisfactory logical view resulting in incomplete data independence.

(ii) Incomplete generality with regard to engineering entities and relationships. They lacked the features to deal with the complexity and variability of engineering data structures.

It was stated that there was no choice other than incremental expansion because the scope and resources required for complete system design made that approach impractical. The strategy was to design and

implement integrated but manageable subsystems which in turn required data independence and independent program modules.

Other requirements included:

(i) Temporal copies of database for case studies.

(ii) DPLS to provide help to users to follow and control updating so that integrity of the database is preserved in the environment of frequent updates.

(iii) Powerful data representation capabilities.

(iv) Minimum and random disk access such as provided by paging I/O method.

(v) Dynamic loading of programs so as not to restrict the users choice of program.

(vi) A single POL that can be added to or changed with ease.

(vii) Macro function to define common command streams.

(viii) Tutorial help for users and data dictionary for system development personnel.

(ix) Unit conversion for data values.

(x) Default values for data variables.

(xi) Encoding for standard listed items.

PLAN (IBM's Problem Language Analyser) was adopted to support POL definition, dictionary and interpreter functions.

Application subsystems developed under DPLS were to be provided with:

(i) Rules through which the generality of the database, freedom in program communication, and consistency among user interfaces are ensured.

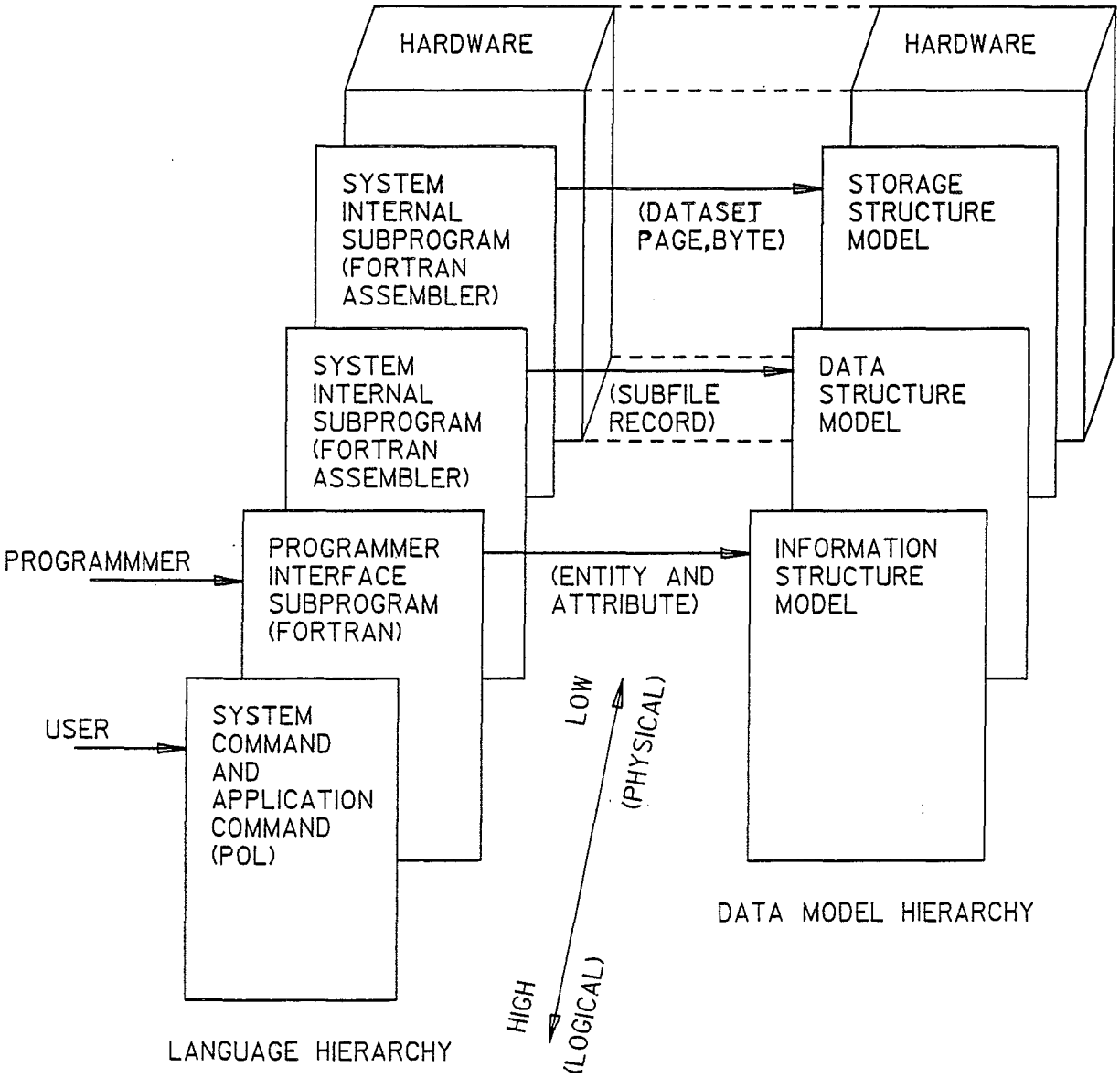


Figure 38. Data Model and Language Hierarchy in DPLS (Source: Tsubaki,1975)

(ii) Tools with which database, programs and user interfaces are developed and maintained at minimum cost and time.

Programs were kept in three levels of load module libraries, namely: command programs; programmer programs; and internal system programs (refer figure 38).

There were three levels to the data model of DPLS:

(i) The entity and attribute concepts are contained within the Information Structure Model (ISM). Users and programmers can access any value (which may be variable in length) by looking at ISM. DPLS provided four methods to express entity relationships.

(ii) The Data Structure Model mapped the logical subfiles (described in ISM) to physical subfiles. Multiple case studies were supported at this level.

(iii) The Storage Structure Model described how the physical subfiles are stored on physical devices.

Reorganisation of the database was claimed to be achieved with a few commands.

The databases were classified into five categories:

(i) System administration and database structure and status.

(ii) Utilities such as tutorial help and Data Dictionary.

(iii) Object database for storing information on a single project.

(iv) Constant database of general reference only material.

(v) Backup database for recovery purposes.

Access to data was determined from the user's access rights, concurrency control and context control (that is, whether another user had locked out the data).

Provisions for integrity such as invalid value check, update flag and update propagation were added. Optional attributes to describe dates and users names associated with stored values were considered. Logging of the command stream was also provided to facilitate recovery.

The data dictionary included commands, macros, source programs, load modules, libraries and relationships between these entities. Various statistics were accumulated to aid system evolution.

Program loading was achieved with the use of a dynamic link function and communication between commands and programs was through global variables (a blank COMMON data area).

An extracted portion of a database could have been produced by copying part of the original database. This extracted database offered an efficient mini-project database.

DPLS also provided "family" management facilities. A "family" was a collection of entities that made up a standard process plant module. The definition of a family module included : its identification and type, internal entities that make up the module (identification and type), entity relationships among internal entities, relationships to external entities and finally default attribute values. Tsubaki (1979) has reported on further details of DPLS.

The criticisms of ICES with respect to portability also apply to DPLS although the DPLS database effort is a significant advancement.

(3) Building Description System (BDS)

BDS was being developed to investigate some of the unsolved issues existing at the time in the structure of large databases for design (Eastman, 1976). It was designed to operate on a minicomputer with very limited memory by today's standards.

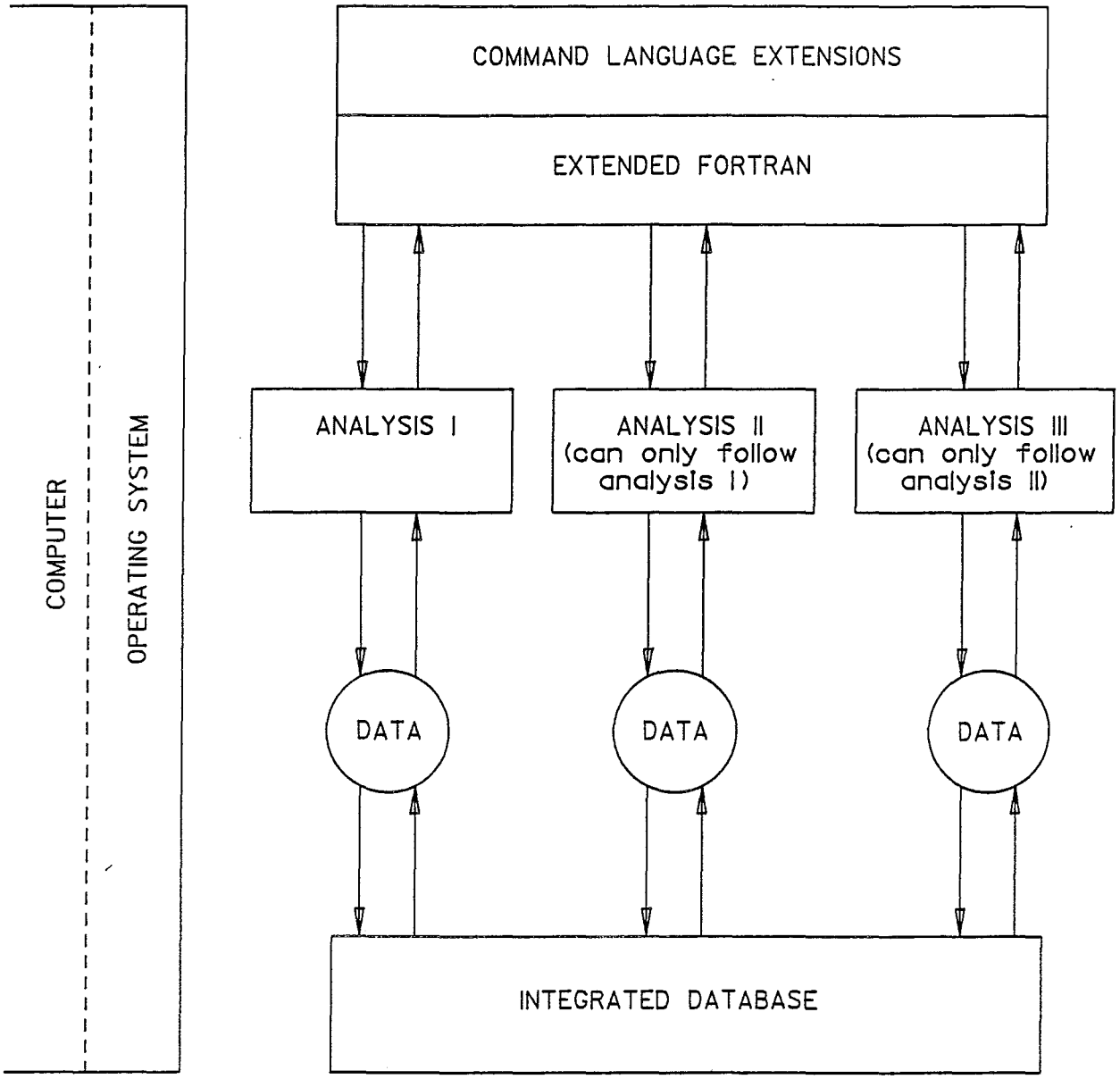


Figure 39. A Second Stage CAD System
(Source: Eastman,1976)

The BDS effort was an attempt to define and demonstrate a database based on non-redundant data. The paper described a database system in which non-redundant data was to be kept and was to be capable of computing the redundant data when required. (Eastman used the term redundancy to refer to both multiple copies of the same data and data which is functionally derivable from other data.) The functionally related data was to be reduced to independent components. When a particular analysis required data to be organised in a specific way (including unit of measure transformations) the data was to be computed from the stored data, manipulated then re-integrated into the non-redundant description. Eastman termed this the third stage of CAD systems.

In a second stage system all the data requirements for all the anticipated analyses and for drafting are collected together into a single large database. The information on each conceptual entity and its relationships may have been stored in multiple ways depending on the requirements of the various analyses. An alternative was to map data at specific points in time into their functionally related forms with predefined directions, thus, imposing a fixed sequence of design stages (see figure 39).

Eastman claimed most analyses dealt with a small set of non-spatial properties that are fairly disjoint from those considered in other analyses. In contrast spatial properties are used in most analyses. These statements are not true for wagon design: stresses, loads, internal forces and moments, displacements, accelerations, material properties, structural stiffnesses, mass properties are neither small nor disjoint compared to spatial properties.

Functional relations are a form of redundancy which will introduce consistency and integrity problems unless mechanisms to deal with it are introduced. However designing a non-redundant database as proposed by Eastman introduces the following problems:

(i) Identifying what exactly are the independent components to be stored in the database.

(ii) Slowing response time by moving the problem of redundant data to redundant processing, impacting computing resources.

(iii) How do you computerise the mapping of the human decision maker?

(iv) How are trial or experimental changes (multiple candidate solutions) that are not for release to other groups of the organisation catered for?

The consistency/integrity problem is not necessarily one of removing all forms of redundancy. The issues of propagating changes, identifying and relating versions must also be addressed. That one set of input data produced one set of output is still true if another set of input is created. What has changed is that there is no derived data (till calculated, or decision made, etc.) when new input is created (including the case of when the new input is created by editing the old input).

(4) Corporate Graphic Systems (CGS)

General Motors (GM) is a very large company with more than 10,000 GM and supplier designers. It has put a substantial effort into CAD/CAM over the past 25 years and has passed the point where more than one in three parts components are on CAD.

The latest cycle of development is a five year integration project based around CGS (Anon, 1984). CGS's main features were its menu processor, database manager and system interface. They define an integrated system as one which provides an individual user, sitting at a single terminal, access to all needed data and all needed applications in an easy and natural manner.

GM expected the project to be showing a positive return on investment before the project was due to finish. Projected gains included: increased productivity, improved potential for creativity, innovation, faster response to market demands and improved quality.

GM wished to feel unconstrained with regard to what was available from vendors, but their plan for expanding CAD/CAM was constrained by the investment, not just in hardware and applications, but also in trained staff and in existing manual and computer based drawings. Training and experience, as well as data were recognised as real investments and real costs were incurred when discontinuities were introduced into training and data conversion. Their path then was an evolutionary one with emphasis on standards such as Initial Graphics Exchange Standard (IGES) and reduced proliferation of suppliers.

CGS, their own sculptured surface program, was to be the core of the integrated system. But GM intended, whenever possible, to buy applications such as CADAM, CATIA, DRAM, ADAMS, SYSCAP and GEOMODS so that their resources could be concentrated on development of overall system architecture.

The five phases to the integration plan were:

(i) The data exchange phase using IGES format files to exchange data between multiple systems. This was being achieved with reasonable success.

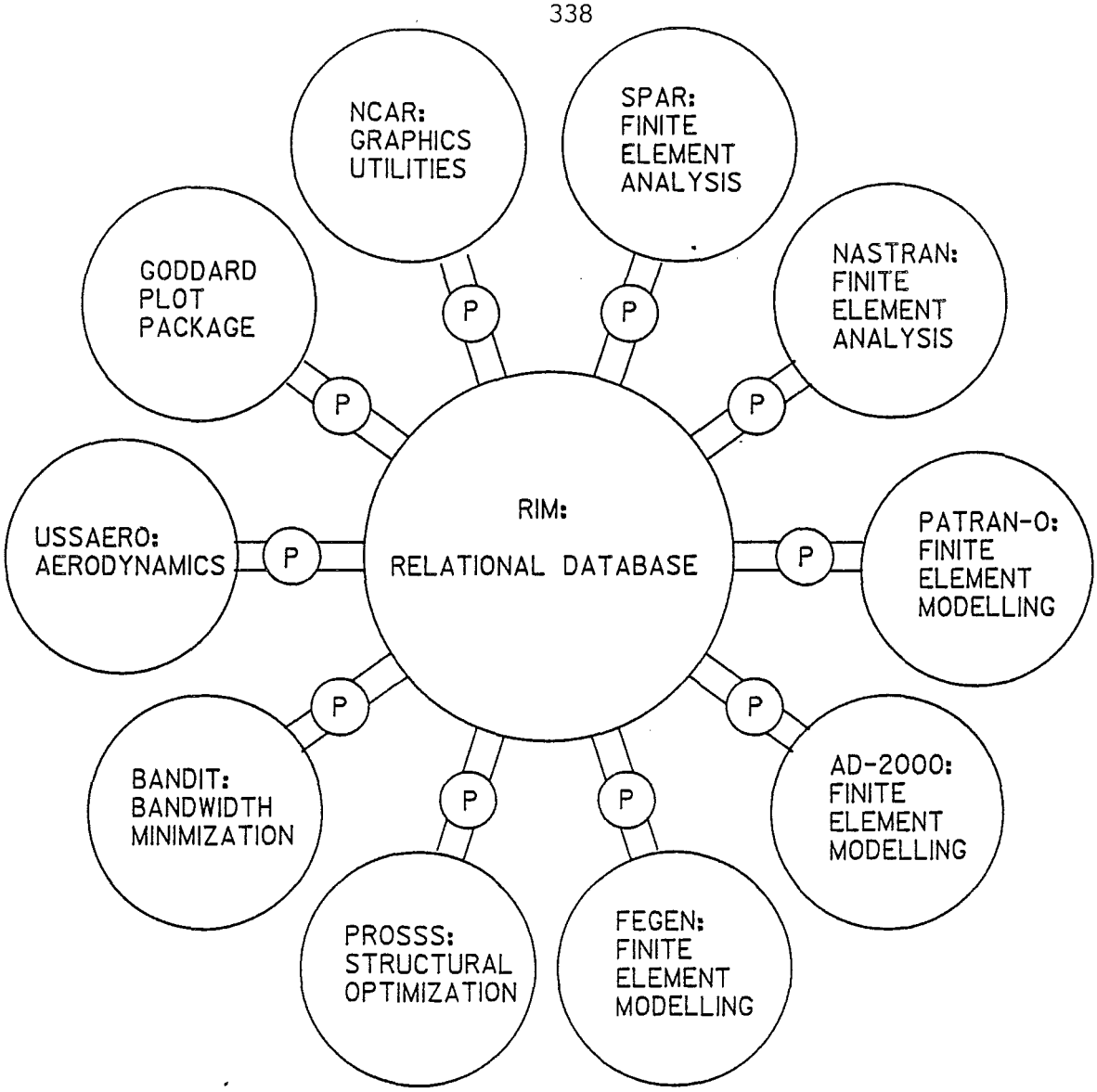
(ii) Graphic console phase enabling a user to access different software systems easily and naturally, with each package retaining its individual features.

(iii) Database phase featuring a single console, many application packages but common database access to all information in CGS and vendor systems. CGS could then use selected vendor data entities for specific operations. CGS or vendor systems would run as before but would be able to access each others data conveniently and interactively (without any IGES-type translation).

(iv) Subsystem phase involved incorporating the vendors application software into CGS as subsystems but with the vendor to continue all development and maintenance.

(v) Full system development was to incorporate in CGS the application subsystems in such a way that the CGS man machine interface replaces that of the vendors. This was to enable systems to be interactive at the operator level.

GM was strongly committed to an "enterprise-wide" database approach encompassing more than one database, more than one computer, and more than one vendor. GM judged IBM's DB-2 relational database to be the best candidate. GM also was planning to acquire finite element modelling and analysis systems which will work directly with the mathematical models in the CGS database. Automatic generation of meshes for finite element analysis was also being pursued. GMSolid, a hybrid Constructive Solid Geometry and exact Boundary Representation solid modelling system, was being developed. GM was also actively involved in off-line robotics programming, computer aided die design and



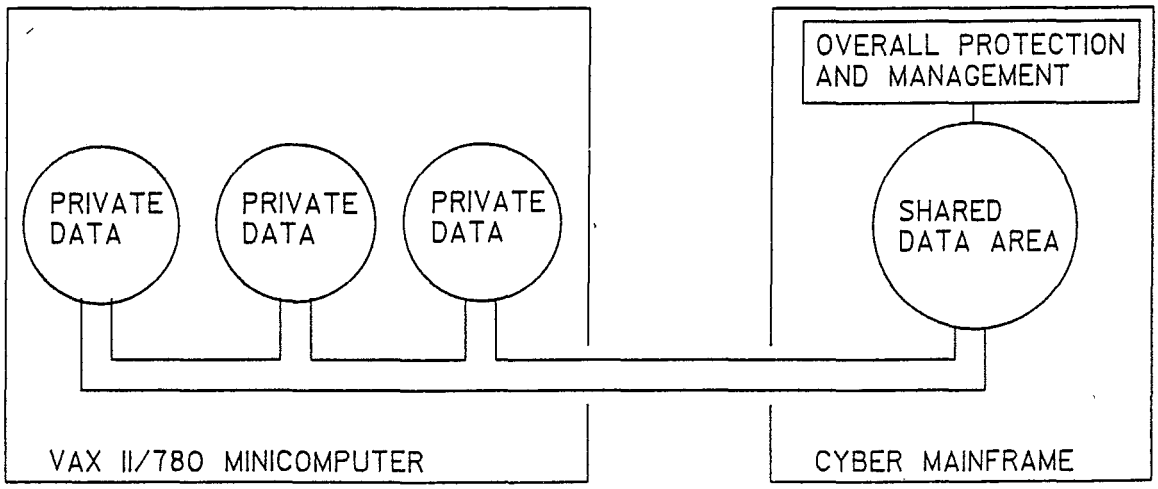


Figure 40. The PRIDE Database Architecture
(Source: Fishwick and Blackburn, 1983)

machining, and a candidate multivendor communication standard (namely MAP - Manufacturing Automation Protocol).

(5) PRototype Integrated DEsign System (PRIDE)

PRIDE was being developed to improve the integration of design and analysis data at the NASA Langley Research Centre as a part of a joint American industry-government program called Integrated Programs for Aerospace Vehicle Design (IPAD) (Fishwick and Blackburn, 1983). PRIDE consisted of a menu executive, a number of application programs, database preprocessors and postprocessors, and a relational DBMS - Relational Information Manager (RIM), (refer figure 40).

Over the course of their aerospace projects, several application programs have been used, because they claim rarely does one program satisfy the needs of every engineer for a given task. Thus data generated from various programs had to be integrated in order to promote a user friendly, interactive design environment thereby increasing probability of a successful project.

Several problems had to be overcome:

(i) Standards for data exchange formats did not exist. IGES was regarded as a definitive first step but much work remained in classification and standardisation of nongeometric attribute data and their relationships with geometric entities.

(ii) Integration capabilities of procured software were generally inadequate. They found some input/output processors required a great deal of unnecessary programming simply because of the application program's inadequacies with regard to data integration.

(iii) Hardware devices from different vendors are often incompatible.

The engineering discipline was regarded as different from the transaction driven, real time environment such as in an airline reservation system where an agent must have real-time access to the most up-to-date transactions. The PRIDE prototype was to investigate a database design which was claimed to be more suitable.

The engineer was to be provided with a private personal database in which initial work was to be performed, and later certain results would be shifted to a shared data area. The shared database was to contain all the data necessary to form a model of an entire aircraft. This data area was to be controlled as integrity was important. The private database was to be flexible and generally associated with specific functions. The engineer was to be able to enter the PRIDE system at any point during the design process because each of the pre- and postprocessors was to access one private database. The use of private and shared databases was viewed as one mechanism for networking databases over many computers. The transfer of data between shared and private database was to be via ASCII (American Standard Code for Information Interchange) formatted files.

The use of a global program network, that is, all programs accessing and storing data in a single database, was advocated, as opposed to many translators (called pre- and post-translators) between individual programs. It was envisaged that as new programs were added they should gain immediate access to the data of all other programs with only one pre- and postprocessor having to be developed. A single interface would then be presented to users and programmers. Many

application programs would still have local data areas for local data manipulation. PRIDE used this concept to relate application programs to a private database.

One advantage claimed of the inter-application pre/postprocessor approach is speed. Data would not have to be channeled through a global DBMS and would be specifically formatted for the receiving program.

A relational database was chosen because flexibility and ease of use outweighed any reduction in response speed compared to the network approach. To maintain integrity certain rules or constraints would be applied. The high degree of data independence was thought to be valuable in an engineering design environment where data and schema are constantly in a state of flux. In particular RIM was chosen for the test-bed partly because it incorporates double precision, vector and matrix data types which may be fixed or variable in length.

However restricting the medium of data transfer to ASCII formatted files (as in PRIDE) is too limiting. It is a suitable means of transferring data to and from proprietary applications whose source code is for practical purposes unattainable and which does not provide a programming interface. However when feasible it is more computer resource efficient (and consequently faster) to develop a program to directly transfer the data between application and database. In addition, a DBMS may have tools specifically supplied to produce an external view, use of which could be more efficient and provide better data independence.

(6) Building Services CAD

In Aish's paper (Aish and others, 1985) on an "integrated CAD system" for building services engineering, they described the generalised data flow through various programs used in building services engineering. They claimed the designer through idealisation (or the level of detail in the model) of the real building and with choice of program has full control over data flow and decision sequence, demonstrating a wide range of courses of action depending in part (at least) on the circumstances. Both computerised approved hand calculation methods and more extensive computer-based methods were available.

Initially a few discrete application programs were available to their users. Users accessed these directly through the computer's own operating system. It was found that this method of access created a significant training hurdle for users and also required continuous supporting effort from the computing staff.

The objective was to provide computational facilities to assist the building services engineer to perform his function. One of the main features was to provide flexibility in the way the engineer used different system options and mobility of data between these options enabling early progression to more complex variations derived from initial data.

As well as extending the range of engineering programs available, they directed resources toward the development of an effective user-computer interface. More specifically this interface was to incorporate:

(i) A series of menu options tolerant of faults made by the user.

(ii) Data displayed and edited within a set of formatted data modules. Descriptive captions explaining the function or units of each of the data items were to make the data more comprehensible to users.

Programs with minimal volume of input and output data had a spreadsheet type screen. This would enable editing of data in the context of other input and previous output. For the programs that require considerable amounts of data, structured submenus were adopted to act as directories to the data modules. Another feature was a tutorial mode that could be switched off progressively as familiarity with the system increased.

The user logged into SIMCOM (SIMplifed COMmand system) from which the user could select application programs and important housekeeping functions normally available in the operating system. After selection of an application, its main menu was displayed.

MODIT was the interactive data preparation system in which data was created, edited and vetted (against valid range checks) by means of "forms" style screens. In the tutorial mode additional prompting occurred. OUTSET enabled users to select any page of output for display. Each user generated one or more output files unique to each user, integration was through SIMCOM, OUTSET and MODIT programs.

The system was written in FORTRAN77 except for the hardware, operating system and terminal dependent areas. New applications (with the appropriate range data, message file, form definitions) could be added without restriction it was claimed. It was also claimed that this defined software development environment facilitated consistent structure and conventions reducing time and complexity of user training and simplifying development and maintenance.

In the opinion of the author this system did not tackle data integration and data management facilities were weak. The user interface seemed well suited to the occasional user but it did not provide the experienced full time user with the appropriate interface (for example, a command language).

(7) Commercial CAD Systems

The development of drafting/design systems has become the focus of many companies in recent years. Most of these systems were originally developed from products heavily graphics oriented, and aimed at reducing the drafting cost which is a major component in design projects. These drafting systems provide a means for entering, storing, recalling and manipulating 2D or 3D graphical patterns.

The implementation effort involved in these systems is often considerable with hundreds of man years invested and the systems are often sold for hundreds of thousands of dollars. They have been implemented on hardware from custom developed hardware through to standard general purpose machines.

Although the packages are centred around the creation and editing of graphics, including 3D surface and solid representations as well as working drawings, their features include:

(i) Basic analysis capabilities including mass and sectional properties.

(ii) Visualisation of objects from any angle such as shaded colour, perspectives, and hidden line.

(iii) High level graphics programming language which can permit the programming of any of the functions available interactively including man machine interaction.

(iv) Automatic or interactive creation of NC part programs off geometry previously constructed and graphical simulation of the same.

(v) Group technology classification and retrieval subsystems.

(vi) Arbitrary sectioning of geometry and automatic interference checking.

(vii) Command language and/or tablet based menu and/or screen menu.

(viii) Interactive graphic development of finite element models from which finite element analysis can be run.

(ix) Analysis and display of physical test results.

(x) Technical office functions such as word processing, spread sheeting, equation solving, project management, business graphics, electronic notice-board and technical publication.

Some vendors purport to sell Computer Integrated Manufacture (CIM - the whole manufacturing system (design, analysis, etc.) integrated and highly automated by means of computer systems); a claim that needs to be approached cautiously in the light of current offerings and research projects.

GE CAE International is an example of a company offering products addressed at the mechanical design and manufacturing market with strong graphics and analysis capabilities. Their products included : SDRC GEOMOD, a solid modeller; SDRC SUPERTAB, a finite element modelling package; SDRC IMP, a three dimensional mechanism analysis program; SDRC SUPERB, a general purpose finite element analysis program; SDRC MODAL-PLUS, an interactive package to collect, analyse and display data from controlled excitation tests; SDRC FATIGUE, a fatigue life prediction program; SDRC SYSTEM DESIGN, a solid modelling program for manipulating

and assembling individual components; SDRC SABBA, predicts natural frequencies and modes of vibration of systems.

Some companies sell both graphics and nongraphics database facilities. Intergraph, for example, sells in addition to its graphics software, a CODASYL network type database management system. This DBMS incorporates many of the features found in other commercial nongraphics DBMSs such as: host language interface (HOL); data definition language and compiler; a data entry screen design and forms language; report generator; data manipulation language; a screen oriented, columns and rows method of updating and viewing the database. The nongraphic attribute data is not stored in the design file but in one or more network type databases, and yet is directly accessible from interactive graphics. A database can be attached to a graphics file and linkages of various types established between the graphic elements and the nongraphic attribute data. The attribute data may be created, edited and reviewed from the graphics session. Selected attributes can be formatted and displayed in the graphics file for annotation purposes. These display attributes may be updated automatically when changes occur in the database. Attribute search criteria can be used to select graphic elements or the user can specify the graphic elements to be reported on.

Intergraph have used their DBMS and graphics software in systems developed for particular application areas such as plant design. The Plant Design Piping and Instrumentation Diagrams Module has had three types of databases designed for it.

The Reference database was to be the repository for job specifications, industry design codes, vendor catalogue data, commodity

libraries, past designs and computing practices. It was to provide project management with the means of controlling consistency in the various disciplines associated with plant design. It was to enable definition of user access privileges; units of measure; part identifications and descriptions; project organisation; definition of drafting standards, nongraphics attributes associated with items, and specifications used with items; symbolic equivalents of items; presetting of command options; report formats; user messages; and definition of graphics presentation of items.

The Task database was to be the working environment for each discipline. The user could retrieve data from the Reference or Master databases and at any time generate working drawings and reports. Each graphics file was to be a unique partition of the project. The technique of propagation would allow minimum loading of attribute data into the database at interactive placement time and bulk loading after the drawing was completed. Thus propagation, it is claimed, permits design rule checking before approval as well as more efficient use of computer resources and faster interactive response.

Approved designs were to be sent to the Master database, which was the single source of all lists, reports, drawings and other documents.

Intergraph has aimed to produce an "open" system and indeed many of the basic interfaces and tools are provided including facilities for adding simple commands and menus.

In theory, drafting and graphical presentation are just other application programs. The data needed could also be integrated with the data required for analysis but closer investigation would reveal that apart from the requirement for graphics capable output devices, speed is

important for productivity (which has implications for the data structures and the processes which manipulate them). In addition graphics operators would be more familiar with the subset of commands used for graphics than the designer with the wide ranging design functions (and this will influence the user interface design). Intergraph's products are interesting to note in this context.

(8) NOMAD

NOMAD described by Martin (1982) is an example of a proprietary "Fourth Generation" information manager used in the business and administration worlds. Other similiar systems include ORACLE, MAPPER, ADABAS, IBM STAIRS and SYSTEM 2000.

NOMAD was designed for storing and manipulating data in a database. Its DBMS has facilities for report and graphics generation and high level language programming.

NOMAD has both procedural and nonprocedural statements. Martin contends that the syntax is easy to perceive at a glance and it is easy to understand and modify another user's programs. Conversational mode contains both nonprocedural and procedural statements which execute interactively. For example, one command line can generate a report with automatic formatting, generation of page numbers and column headings. Routines can be stored and recalled at a later date.

Data can be stored in relational or multilevel hierarchical or hybrid form (which contains both relational and hierarchical structures) offering the advantages of relational flexibility (and dynamic access paths) and efficiency when following hierarchical paths.

The DBMS also features concurrency control and locks for multi-user simultaneous shared access, protection from system failures and automatic audit trails. The schema definition includes: column headings; alternate names for fields (aliases); variable length field definition; derived field types (automatic derivation of value); primary key definition; limits to range of values or to be a member of a set of values; alphanumeric mask ensuring conformance to specified character pattern; and member items referring to an entity which must be a valid entity.

Updating does not destroy the old data till new data is correctly and completely in place. There are a number of database record access methods including access to any fixed length sequential file. Data can be loaded from sequential files, or the user can be prompted for data items individually, or formatted data entry screens can be defined. A set of values may be updated with one command. Query facilities include row/column totals, subtotals, averages, standard deviation, minimum, maximum, count of values, etc. Case studies or "what if" studies can be performed without altering the original values in the database. Textual, reports stored in NOMAD can have values inserted into them from the database.

As well as passwords, and read and update access restriction definition, security measures allow the definition of logic to control update and access restrictions.

Line, scatter, bar and pie charts can be plotted or displayed in colour.

Fixed and variable length, multi-dimensional arrays are valid data types. Time series are treated as a special type of array with various

functions for handling them being provided. Array manipulations are included.

The procedural portion of NOMAD contains statements which express logic. Operators which perform arithmetic, comparison, logic operations, table lookup, looping, conditional branching, condition and interrupt handling, string manipulations and statistical functions are provided.

NOMAD presents a good end user environment - easy to learn and use, yet powerful and flexible.

3. CONCLUDING REMARKS

It is evident that the full potential of the digital computer has not yet been fully exploited for wagon design. Because of the size of the undertaking, no system exists which does not have weaknesses in some area. World-wide efforts are continuing to strive for the "best" computer-aided design environment. There is some agreement on what is required, but the systems being developed are specific to a company or discipline and are therefore different in intent. However, there is no generalised system with which to build a computer aided wagon design system as specified in the earlier chapters. Standards have only just begun to appear in some areas.

NZR must then select a candidate system or systems to modify and enhance, as it can not afford the development of a complete system because of its small scale of operations. The availability of details of the selected system(s) data structures and program interfaces is therefore important.

4. REFERENCES

AISH, R. and others. (1985) Integrated CAD Development for Building Services Engineering. Computer-Aided Design, 17(4) : 179-190.

ANON. (1984) Integrated CAD/CAM : the General Motors approach. Automotive Engineering, 92(10) : 54 - 59.

BALLING, R.J. and others. (1986) Methods for Interfacing Analysis Software to Optimization Software. Computers and Structures, 22(1) : 87-93.

BANAHAN, M. AND RUTTER, A. (1983) The Unix Book. New York, John Wiley & Sons. 218p.

EASTMAN, C.M. (1976) Databases for Physical System Design : a survey of US efforts. Proc. CAD 76 - International Conference on Computers in Engineering and Building Design, London. Guildford, IPC Press. p1-10.

FISHWICK, P.A. AND BLACKBURN, C.L. (1983) Managing Engineering Data Bases : the relational approach. Computers in Mechanical Engineering, 1 (3) : 8-16.

GAGGERO, G. (1981) WESPOL - A Programming Language for POL Development. Proc. of International Conference on Computing in Civil Engineering, New York. New York, ASCE. p.688-699.

LEROY, Z.E. AND GREEN, D.B. (1981) ICES Concepts - A Modern System Approach. Proc. of International Conference on Computing in Civil Engineering, New York. New York, ASCE. p.89-107.

MARTIN, J. (1982) Application Development Without Programmers. Englewood Cliffs, New Jersey, Prentice-Hall. 350p.

TSUBAKI, M. (1975) DPLS: database, dynamic program control and open-ended POL support. Proc. Workshops on Databases for Interactive Design, Waterloo, Ontario.
p. 146-160.

TSUBAKI, M. (1979) Family Generation in Integrated Engineering Systems. Computer Aided Design, 11(3) : 136-141.

CHAPTER XV

DESIGN AND DESCRIPTION OF THE PROTOTYPE INTEGRATED WAGON DESIGN SYSTEM

The computer programming and a database definition in partial fulfillment of the integrated wagon design scheme specification are described in this and the following four chapters. The overall functions and structure are described before those of the command interpreter, run-time interpreter and database respectively, and finally a sample session is presented.

1. PURPOSE AND SCOPE OF IMPLEMENTATION PHASE

The overall objective of this phase of the project was the formulation and testing of an integrated system design upon which future system designs and implementations could be based.

As the first formal study of integrated computer aided wagon design within NZR, it was intended to provide an initial foundation from which further development could proceed. It was also to provide something concrete for NZR to evaluate, an exercise in which the practical experience gained could be used to guide future development work and/or selection of commercial systems. The implementation should prove the feasibility of the specification and enable a more rationally based evaluation of costs and benefits. The implementation was not to be regarded as a production system.

The overall form was to be similar to that expected of the full system, but only a practicable subset of functions (in terms of

available resources) were to be included in the prototype, and yet enough to meet the objective. Implemented were an extendable interactive command language, command and instruction interpreters; definition and use of a database through a general purpose database management system; support for trial cases and experimentation, stored relationships between data and variables; instructions to perform finite element analysis; suspension of execution to permit solution of other problems when dependent variables do not exist for the current command and resumption when the wagon designer wishes. Specific features were derived mostly from the works of Hunt (1983), Fenves (1976), Holtz and Fenves (1980), and Tremblay and Sorenson (1982).

For reasons outlined in chapter I, section 3(1), this prototype system did not address draughting or graphics functions. No additional facilities for extending the range of commands and data entities, or for tailoring the user's environment have been implemented. Language structures were limited and only the basis for programmed transition forward to or backward from the independent variable(s) has been realised.

2. OUTLINE STRUCTURE AND FUNCTION

(1) Functions

(a) Models and Methods. The prototype integrated design system provided a flexible aid to the solution of problems in a network of variables. The problem was described by a subset of variables with one or more unknown(s). A method was used to produce the unknown(s) (that is, solve for the dependent variable(s)). The relationship between the

dependent and independent variables and the method was called a model. For example, the STIFFEN model related structure definition, stiffness matrix and method instances.

The model relationship did not restrict the choice of dependent variable(s), rather valid methods for dependent variable(s) in models were stored in the Data Dictionary. So in the example above, the structure definition could be derived from the stiffness matrix if there existed a method in the Data Dictionary. The Data Dictionary also contained default methods as a number of methods may be valid for the model, dependent variable(s) relationship (banded symmetric Cholesky decomposition and Gaussian elimination in the solution for displacements in a finite element model). The input of values by the wagon designer was a creation method available in all cases.

Run-time integrity checks were performed, for example, does a dependent instance already exist which was formed by the specified method and with the specified independent variable instances; is the specified method a valid one for the model and dependent variable(s); do the specified independent variables instances exist. If the independent variables did not have values defined in the database the command did not abort, rather it suspended execution until the wagon designer was ready to return to it.

By storing how values were arrived at, that is, using which variables (as defined by the model entity), which values of these variables, and what method (the model instance), as well as dependent and independent variable names, and date of creation, the wagon designer could navigate around the network of variables. The relationships between the data values were stored by means of these relationships

between the variable instances (that is, the model instances) thereby providing a network of data values. Other than the integrity checks, available computer methods, and variable entities, the wagon designer was free to determine the direction, in solution of a chain of problems. With this system, the system design problem has become one of what to describe to the computer system about wagon design rather than how.

(b) The Command Language. The prototype integrated design system ran in an interactive mode where commands were immediately executed. A session consisted of any number of commands. The commands took the general form of "verb expression" where verb is the operation to be performed on the expression containing nouns or identifiers and values. The command language was designed to overcome the lack of portability and unnaturalness to wagon designers of the modern interactive languages.

The assignment command assigned values to user defined or system defined variables. The variable type could have been character string, numeric (all treated as real numbers), or logical. No array types were implemented. These variables were global in scope, only active during a session and initialised at the start of a session (as opposed to the variables stored in the database). Expressions could have contained arithmetical, logical, and relational operators and built-in trigonometric, logarithmic, and square root functions. The write command displayed constant and expression values to the designer. The QL command initiated the DBMS Query Language.

The solve command calculated and then stored the value(s) of the dependent variable(s) in the database. It allowed the wagon designer to

select the method and independent variable instances. If these were not supplied the wagon designer was prompted with default values. At that stage the wagon designer could suspend the command by executing any other command or quit that command. The return command resumed a previously suspended command. The default instance names of the independent variables were taken from the current values of the system defined temporary variables. An example of the solve command follows:

SOLVE STIFFEN FOR 'BTK VERSION 3' STIFFNESS WITH 'BTK VERSION 3' STRUCTURE USING BANDED STIFFENER;

(2) Outline Structure

The prototype integrated design scheme consisted of three parts: the command interpreter, the run-time interpreter and the database system. The main routines performed initialisation and invoked the parser.

(a) Command Interpreter. The single pass command interpreter consisted of the scanner, parser, table handler and code generator routines. The scanner was invoked by the parser to read the next unit of source text and return a token representing this lexical unit to the parser. The parser performed syntactic analysis of the command being interpreted. The routines of the table handler looked after insertion and look up of symbols in the symbol table. The symbol table was used to store information about identifiers such as type (numeric, logical, or character string) and scope (user defined or database entity). The code generator performed the context sensitive syntax checking, analysed the semantics of the program, and generated the appropriate code to

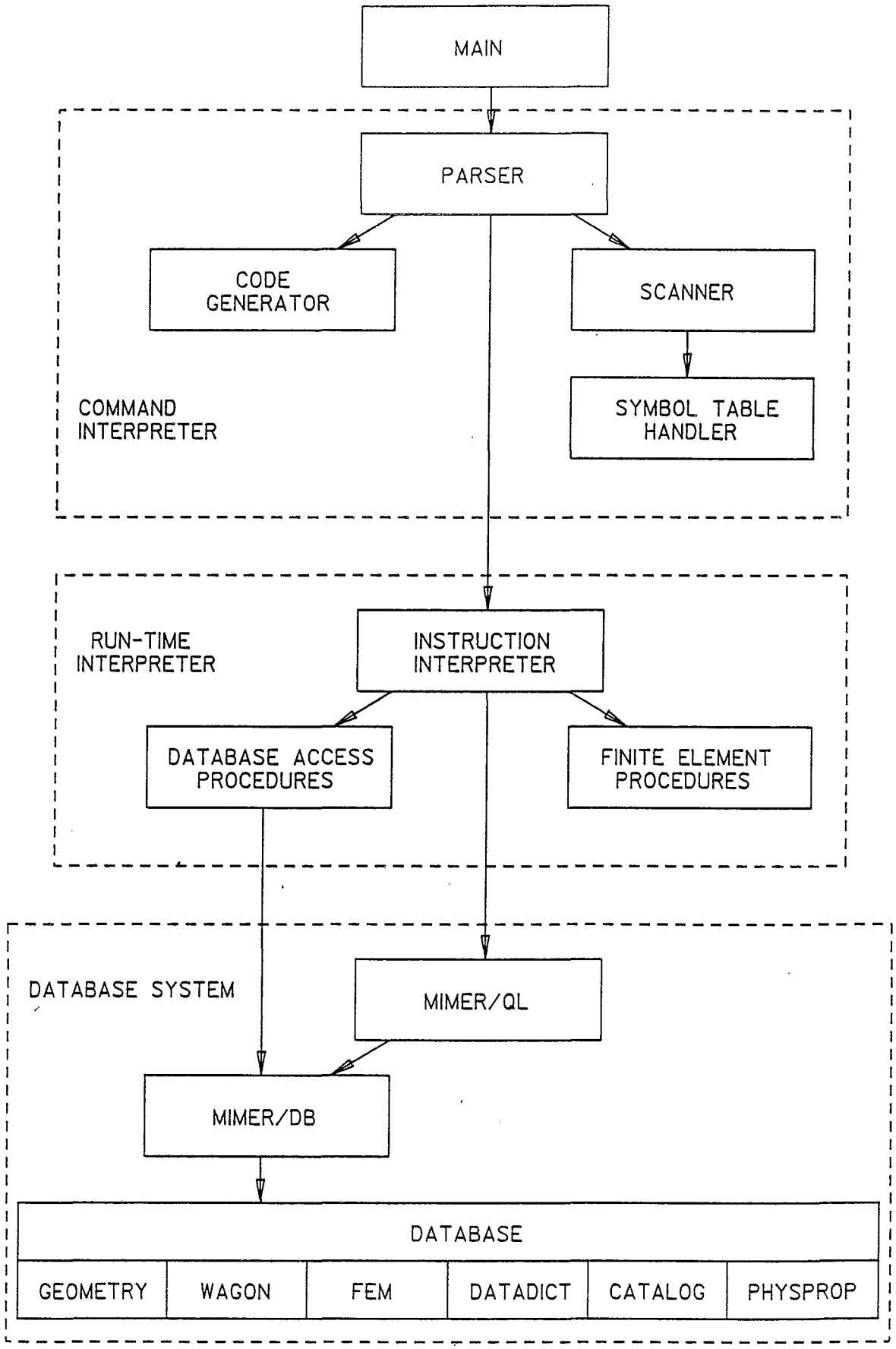


Figure 41. Prototype Integrated Wagon Design System Architecture

perform the required operations in order to execute the command. Figure 41 illustrates how the procedures invoked each other. The parser invoked the scanner and code generator routines. The scanner routines invoked the table handler.

(b) Run-Time Interpreter. The run-time interpreter routines comprised the interpreter that executed the code generated by the command interpreter. In addition to error and run-time stack handling routines the instruction interpreter invoked three routine libraries: the DBMS Query Language, the database access and update routines, and the finite element procedures. The database routines invoke the host language interface of the DBMS and perform: Data Dictionary update and accessing; retrieval of model definition, default method, dependent and independent identities; check method, model and solution definitions; check presence of dependent instance; create dependent instance; load data arrays to be used in the finite element routines from the database; and insert finite element routine results into the database. The finite element routines invokable were those to number degrees of freedom and find the semi-bandwidth, form the banded finite element stiffness matrix, and form the force vectors for a number of load cases. Other finite element routines were available but were not fully implemented as an integral part of the prototype design system.

(c) Database System. The DBMS consisted of a database handler and an application program which interpreted the query and command language. The databanks implemented were:

(i) Geometry: a description of geometrical features and geometrical families of parts.

(ii) Wagon: part, assembly and wagon data and relationships.

(iii) FEM: finite element data and relationships.

(iv) DataDict: dictionary of databases, tables, attributes, interpreter variables, valid solutions, default methods and models.

(v) Catalog: geometrical reference data including properties of standard sections.

(vi) PhysProp: reference data on material properties.

3. IMPLEMENTATION

(1) Implementation Phases

The implementation by the author commenced in June 1983 with the first phase - the development of a linear statics finite element program. The finite element program was developed to provide modifiable analysis source code relatively cheaply and to provide the author with programming experience. The program calculated displacements, forces and moments, and stresses for constant cross section, two noded beam and eccentric beam elements, and spring elements. Loads could include nodal forces and moments, prescribed displacements and rotations, concentrated point loads, uniform and linearly distributed loads on beam elements, body forces, and load case combinations. Restraints could be applied in the global axis system. Free format input of data was developed. The programming was performed for a PRIME 750 running PRIMOS revision 19.1.

The second phase began with the development of the geometry, wagon and properties databases in November 1983. The relational DBMS MIMER 3.1 (Savant Enterprises, 1983) was used. MIMER was running only on the Burroughs 6900 machine, and therefore subsequent development was performed on this system.

MIMER is a suite of products aimed to speed up application development. The full range of products included: the database handler - a relational database management system for creating and maintaining the database, controlling an active dictionary of data items and interfacing with conventional systems; a query language - an english type language suitable for end users with commands for accessing, building and extending the database; a prototyping language and program generator; a data entry and query screen handler system; an information retrieval system for large data and text search applications; utility programs to execute frequent operations on the data base.

In addition to the basic database facilities, MIMER provided additional features, including:

- database recovery management;
- automatic concurrency control;
- authorisation mechanisms;
- data independence (high level of physical independence but rather less logical independence);
- dynamic database definition;
- tuning and usability features (such as secondary indices, control of display, direction of output and on-line help information).

In July 1984 the third phase commenced with the implementation of a subset of the GAUSS language (Tremblay and Sorenson, 1982) followed by the extension of the database, command interpreter and run-time interpreter to include finite element entity models, methods and variables. Development finished in February 1985. All implementation phases ran concurrent with other activities such as draft thesis preparation.

(2) Encoding Language

Because there was considerable uncertainty as to which computer system the prototype design system would be installed on for Railways Corporation evaluation, language portability was regarded as the most important issue in selection of encoding language. The wagon design system was written in a compatible type of FORTRAN IV and compiled using Burroughs FORTRAN 34.750.

(3) Storage and Memory Requirements

The command interpreter and main run-time interpreter routines were written in approximately 6500 lines of source code with an estimated core storage requirement of 1.4MBytes, approximately 5% of which was program code. The database accessing and updating library routines were written in 7300 lines requiring an estimated 550KBytes of core storage, 20% of which was code.

The finite element library routines extended over 4200 lines and were estimated to occupy 61KBytes of core storage, 60% code (the arrays were declared in the run-time interpreter). A general mathematics library occupied a further 500 lines requiring an estimated 4KBytes of core (again array sizes were declared in calling routines). The finite element array sizes were allowed problems up to 200 elements, 200 nodes with a maximum bandwidth of 25, and 10 load cases.

No attempt was made to optimise space requirements, but it is clear that a virtual storage operating system will be required for the program and data of the complete system.

The Geometry databank with some 540 rows of data occupied

approximately 107KBytes. Wagon with approximately 800 rows occupied 110KBytes, Catalog with 170 different section sizes occupied approximately 46KBytes. FEM with 4 load cases (7 nodal loads and 2 UDL's defined on a total of 6 elements), a 50 beam element half model (including 4 non standard sections) and its assembled stiffness matrix (semi-bandwidth of 78, 216 degrees of freedom (32 of which were constrained)) occupied 209KBytes. The Data Dictionary occupied approximately 100KBytes and included information on some 90 tables with a total of 510 columns.

4. REFERENCES

FENVES, S.J. (1976) Formal Representation of Design Requirements. In SAUL, W.E. and PEYROT, A.H. ed. Methods of Structural Analysis. New York, ASCE. p.739-755.

HOLTZ, N.M. and FENVES, S.J. (1980) Using Design Specifications for Design. Proc. of 2nd Conference on Computing in Civil Engineering, Baltimore. New York, ASCE. p.92-101.

HUNT, A.A. (1983) Computer Aided Design of the Body Structure of Bogie Railway Wagons. Christchurch, University of Canterbury. (Final year project report : B.E. : Mechanical Engineering).

SAVANT ENTERPRISES. (1983) MIMER/QL- query and command language user manual: version 3:1. Reprinted by permission by The University of Canterbury, Christchurch, New Zealand.

TREMBLAY, J.B. and SORENSON, P.G. (1982) An Implementation Guide to Compiler Writing. New York, McGraw-Hill. 259p.

CHAPTER XVI

DESIGN AND IMPLEMENTATION OF THE PROTOTYPE COMMAND INTERPRETER AND COMMAND LANGUAGE

This chapter describes the detail design and implementation of the man-machine interface for the prototype wagon design system. A subset of the interface specified has been implemented to demonstrate the scheme.

1. DESIGN AND DESCRIPTION OF THE COMMAND LANGUAGE

The provision for different man-machine communication methods (for example, menu selection or question and answer dialogue), renaming of keywords, support for programmable function keys and cursor control devices, set up of terminal characteristics, tailoring the man-machine interface to the terminal characteristics, and the provision for default values was defined to be at a level beyond that of a standard command language. Only the standard command language was implemented. In the prototype design system, the designer sat at an alphanumeric terminal and typed the input.

The command language was based on Tremblay and Sorenson's (1982) LL(1) GAUSS language. LL(1) grammars have been used for defining actual programming languages for several years, they have strong theoretical background and the parsing algorithm for this class of grammars is efficient. To add further support to the selection of this parsing method for this thesis, Tremblay and Sorenson have published details on

implementing this type of parser including PL/I source code for their GAUSS language compiler. Recursive descent compiling would not easily be implemented with FORTRAN IV as it does not support recursion.

(1) A Session

(a) Initiation. Initiation of a prototype integrated design system session was by typing the RUN IDS/W command. On most systems it would be possible (but operating system dependent) to incorporate this command into the identification (log on) phase or to rename the command to something more meaningful and simpler.

(b) Processing. The prototype design system accepted one command at a time, returning responses to the designer as the command was processed. Some of these responses may have been requests for more information, in which case the designer could have input another command suspending execution of the first command.

(c) Termination. The END command closed the session and again could automatically log the designer off the computer system.

In BNF a design session had the following form:

<session> :: = <command>; {<command>;|<suspend> <suspended solve>}
END

(2) Commands

The following is a definition of the commands of the prototype design system language. The syntax of each command is defined in subsequent sections. Commands could extend over several lines but must be finished by a semi-colon (and a RETURN keystroke). The length of each line could not exceed 80 characters. An arbitrary number of spaces

was allowed within a command.

<command> :: = <stop command>
| <write command>
| <solve command>
| <assignment command>
| <return command>
| <QL command>

(3) Control Structures

Of the control structures which allow choice, repetition, and termination in the flow of control of a program, only the STOP control construct was provided to allow termination of command or command stream processing. The syntax of the STOP command was as follows:

<stop command> :: = STOP

Tremblay and Sorenson (1982) also implemented LOOP-WHILE and IF-THEN-ELSE-ENDIF constructs.

(4) Assignment Command

The assignment command had the form:

<assignment command> :: = <identifier> = <expression>

The value of the expression was copied into the variable identifier. The types of identifier allowed were numeric (or real), logical and string. Values of the numeric type had the precision allowed by the machine on which the prototype design system ran. The logical type had either TRUE or FALSE as a value. String-type were are character strings of arbitrary dynamically varying length up to 80 characters long. No aggregate data types were implemented, however

Tremblay and Sorenson had a multi-dimensional array type. All identifiers were global, that is, the identifiers were available no matter where the designer was in the session. At any point in the session no two identifiers (including model and method identifiers) may be named the same. Database model, method and dependent/independent variable identifiers were loaded (and typed) at initialisation. All other identifiers were typed according to the type of the expression in their first assignment. No other declarations were necessary. In all subsequent cases of assignment, the expression must be of the same type as the target identifier. Tremblay and Sorenson permitted target substring and array element assignments.

The following illustrate some simple assignment statements:

```
X = 37*Y+3.4E3;
A = "HELLO WORLD";
FOREVER = FALSE;
```

(5) Expressions

```
<expression> :: = <real expression>
                | <logical expression>
                | <string expression>
```

The following description defines the syntax of the expressions as well as the precedence and associativity of the allowed operators:

```
<real expression> :: = <real expression> + <term>
                     | <real expression> - <term>
                     | <term>
```

<term> :: = <term> * <factor>

| <term> / <factor>

| <factor>

The binary operators just presented have the following meanings:

+ signed real addition

- signed real subtraction

* signed real multiplication

/ signed real division.

<factor> :: = [-] <real primary>

| ABS <real primary>

| ALOG <real primary>

| ALOG10 <real primary>

| ATAN <real primary>

| COS <real primary>

| EXP <real primary>

| SIN <real primary>

| SQRT <real primary>

These unary operations perform the following:

-	negative of real
ABS	absolute of real
ALOG	natural log of real
ALOG10	log base 10 of real
ATAN	arctan of real
COS	cosine of real
EXP	exponential of real
SIN	sine of real
SQRT	square root of real

<real primary> :: = (<real expression>)

| <identifier>

| <number>

The syntax of a string expression was as follows:

<string expression> :: = (<string expression>)

| <literal>

| <identifier>

Tremblay's GAUSS language supported string concatenation.

Logical expression syntax was as follows:

<logical expression> ::= <logical part><relation><logical part>

| <real expression><relation><real expression>

| <string expression><relation><string expression>

| <logical part>

<logical part> ::= <logical part> OR <logical term>

| <logical term>

<logical term> ::= <logical term> AND <logical factor>

| <logical factor>

<logical factor> ::= NOT <logical primary>

| <logical primary>

The <relation> between the expressions was defined as:

<relation> ::= EQ

| NE

| LT

| GT

| LE

| GE

The relations have their usual meanings. For logical and string

operands, only the equality and inequality relations could be tested.

<logical primary> ::= TRUE

| FALSE

| (<logical expression>)

(6) Input and Output

The prototype design system had only one limited output primitive and no input primitive. The output primitive, the WRITE command, allowed the display of the value of identifiers and expressions.

<write command> :: = WRITE <expression list>

<expression list> :: = <expression> {,<expression>}

Choice of output formats was not implemented; formats were coded into the interpreter.

The following illustrates the output command:

WRITE 'THE VALUE OF X IS:',X,'THE VALUE OF Y IS:',3.27*X+SQRT X;

(7) Query Language

The QL command transferred control to the DBMS query language application and on exit, control was returned to the prototype design system. The wagon designer had to go through the QL user identification procedure (or log in) in the prototype design system.

<QL command> :: = QL

MIMER's data manipulating commands were:

INSERT one row into a table.

UPDATE one or more rows in a table.

DELETE one or more or all rows from a table.

COPY a whole table to or from a sequential file.

COPY row or rows from sequential file to a table.

GET data from table(s).

GET data from table(s) and load work table.

PRINT table.

The data definition commands were:

DEFINE a table.

REMOVE a table.

REDEFINE a table by adding or deleting columns.

DEFINE or REMOVE index on column.

DESCRIBE a table to a user.

define or remove a databank.

DESCRIBE a databank.

The database definition command also supplied facilities for defining, removing and displaying users and their access rights. There were other commands to define the environment for a session, show the current environment and display online help information.

MIMER/QL also offered the facility to create, store, edit and execute a set of commands (or a procedure). In addition to MIMER/QL commands, MIMER/QL procedures could have symbolic variables, and input/output to terminal, assignment and control statements.

(8) Solution of Models

The solve command was a significant extension to Tremblay and Sorenson's GAUSS language. The syntax of the solve command was:

<solve command> ::= SOLVE <model identifier> FOR
<entity variable> {,<entity variable>}
[WITH <entity variable> {,<entity variable>}]

[USING <method identifier>]

<entity variable> ::= <literal><identifier>

| <identifier>

This command calculated the value or set of value(s) of an instance of a variable stored in the database according to a predefined relationship between stored variables. (A variable instance could be an identifier of a set of values, for example, a load set may be a set of load cases.) Both the model, which specified the variables and in which was stored the method used to derive the dependent variables, and the variable immediately following the model identifier had to be in the command. The prototype design system could optionally solve for a number of dependent variables. If more dependent variables were calculated by the method than specified by the wagon designer then the prototype design system stored them also.

The variable specification included a string type identifier and optionally a string literal. The string literal was a user definable name for the instance of the variable in the database (for example, the "BTK" structure). The names did not need to be unique. If the instance name was not supplied then execution of the command was suspended until the default was accepted (the current value of the identifier) or another literal was entered (at which point the new literal became the value of the identifier).

Once a solve command was suspended, the wagon designer could enter the information the prototype design system required, QUIT execution of the command, or enter other commands. If other commands were entered, the solve command remained suspended until the wagon designer RETURN'ed to the solve command. The suspended command syntax was as follows:

<suspended solve> :: = <literal>;

| <real>;

| ;

| <method identifier>;

| QUIT;

| <command>;

The number was used to select from a list, and the semicolon was used to accept the default value. Both the QUIT and RETURN commands returned control to the suspended command including to but not beyond the primary level (or the level without any suspended solve commands and akin to the main routine level in procedural programs).

The optional WITH <entity variable> clause in the solve command specified independent variables and their instance names. The system prompted for the names of any independent variables omitted from the command but part of the model and method specification in the database. If the name was not unique then again the command was suspended until the wagon designer selected an instance from the list of instances with the same name or accepted the default selection.

If the method was not specified the wagon designer was again prompted with a default method (as stored in the database for the calculation of the specified dependent variables in the model) if one existed.

Thus the wagon designer had to supply only the minimum amount of information initially, that is, the variable he wanted a value for and what its relationship was with other variables (that is, the model). The wagon designer could obtain information on these from the Data Dictionary by using the query language.

The prototype design system checked that the specified variables were defined in the database and that the specified dependent variables were valid for the method and model combination. If an independent instance was not stored in the database then the prototype design system suspended the solve command and returned a message saying so. The prototype design system also checked before calculating the value(s) for a relationship already existing in the model table between the independent instances and any of the possible dependent instances (as there may be more than one specified by the given names).

The prototype design system stored the value(s) and the dependent(s) name(s), date of creation, source model, and other data as well as the model relationship.

The following are examples of the solve statement (fuller examples are provided in chapter XIX).

SOLVE STIFFEN FOR 'BTK' STIFFNESS WITH 'BTK C' STRUCTURE USING BANDEDSTIFFENER;

This is an example of a complete command that forms a stiffness matrix called BTK from the BTK C structure definition using the BANDEDSTIFFENER method. The relationship between these entities is stored in the STIFFEN model.

SOLVE IMPOSSIBLE FOR RHS;

This command presents the solve command in its shortest form. A RHS is to be determined using the IMPOSSIBLE model.

(9) Lexical Structure

This section describes the lexical structure of the design command language, that is, the symbols which form the basic units of the

language.

<identifier ::= <letter> {<letter>|<digit>}

<letter> ::= A|B|C|...|Y|Z

<real>::= <number part>

| <number part><exponent part>

<number part> ::= <digit> {<digit>}

| <digit> {<digit>}.

| .<digit> {<digit>}

| <digit>{<digit>}.<digit>{<digit>}

<exponent part> ::= E[-]<digit>{<digit>}

<digit> ::= 0|1|2|...|8|9

<literal> ::= '{<symbol>}'

<symbol> ::= <character>|''

<character> ::= <digit>|<letter>|(|)|*|+|,|-|.|/|=|;

Identifiers were of arbitrary length; however, only the first sixteen characters were significant. Keywords were reserved words; thus no identifier could be the same as any keyword. Identifiers, reals, or literals could not be split across input line boundaries (that is, over 80 characters). Embedded blanks were not allowed in identifiers or in numbers. The value of a real could not exceed the capacity of the machine. If it was necessary to put a quote mark within a literal, it was entered as ''. Comments began with C and must be followed by a blank space. They could begin anywhere in the line and extend to the end of the line.

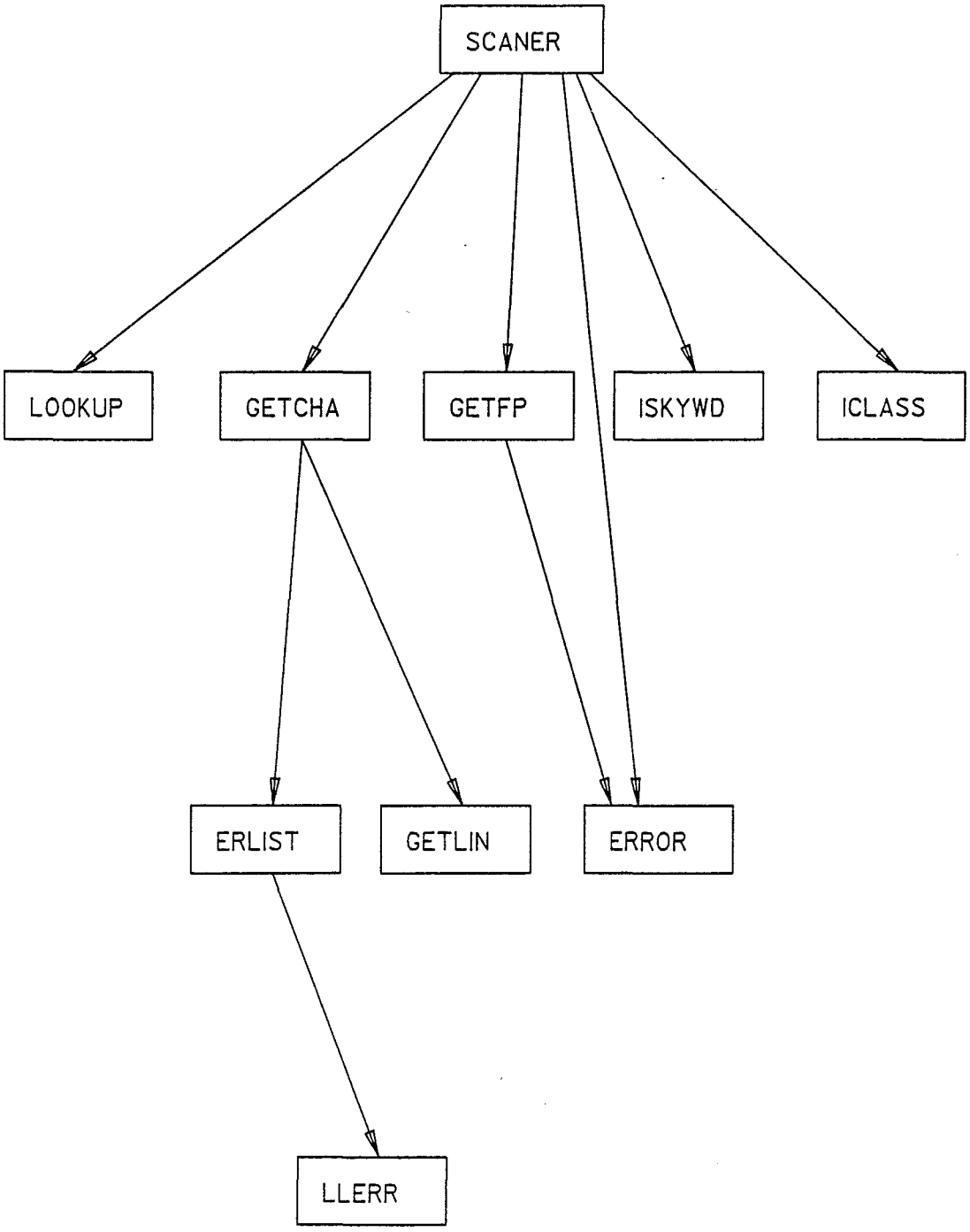


Figure 42. Structure Chart for Prototype Design System Scanner Routines

2. IMPLEMENTATION OF COMMAND INTERPRETER

This section presents the methods and data structures used to implement the command interpreter. The command interpreter read the source commands and generated the appropriate code to perform the command. The run-time interpreter executed this code. Information, requests and error messages were returned to the designer. The command interpreter consisted of the scanner, parser, table handler, and code generation routines. The reader is referred to Tremblay and Sorenson (1982) for more detailed discussion.

(1) Scanner

The scanner read in the characters of the source command and formed them into the basic symbols of the language. This section of the interpreter needed to know about the characters that made up the basic symbols. It needed to know how to form literals and recognise identifiers and reserved words. These basic symbols were represented inside the interpreter by tokens. Thus the basic task of the scanner was that of editing the source commands so as to remove information that was not important to the parser and code generation phases of the interpreter, such as, blanks and comments, and passing on to the parser the tokens representing the source commands. Since there are only a finite number of actions which the scanner could take for any input, the scanner can be described by a finite state machine. A state transition diagram for the finite state machine implemented is presented in appendix I.

Figure 42 demonstrates the calling sequences of the scanner and its

subordinate procedures. The ISKYWD procedure took a given character string and determined whether it was a keyword. The GETFP routine read a real number and converted it from character format to real format. The method used was based on the symbol state table method presented by Day (1972). The transition table used is presented in appendix I. The GETCHA procedure returned the next character in the input stream. Adjacent blanks were merged. GETLIN prompted the wagon designer then read an 80 character line off the terminal. A continuation prompt was written if the command had not been terminated. The GETCHA routine also called the ERLIST procedure which displayed the errors in the command accumulated in the error buffer. LLERR was used to generate error messages for errors which occurred in parsing the command. ERROR was used to insert error parameters into the error buffer. LOOKUP is discussed in a later section. ICLASS mapped the new character into its class and replaced Tremblay's char-class vector.

Other differences in data structures and implementation from those presented by Tremblay are that there was no end-of-file test, no options processing and tokens were integer constants not PL/I macros.

(2) Parser

The process of validating the syntactic correctness of a command stream is accomplished by reading the input symbols and deciding whether the next symbol can legally follow what has already been read. The parser obtained tokens representing input symbols from the source commands by calling the scanner. As the parser analysed the command stream, at certain times it called appropriate semantic routines to generate code. After a semantic routine had completed its processing

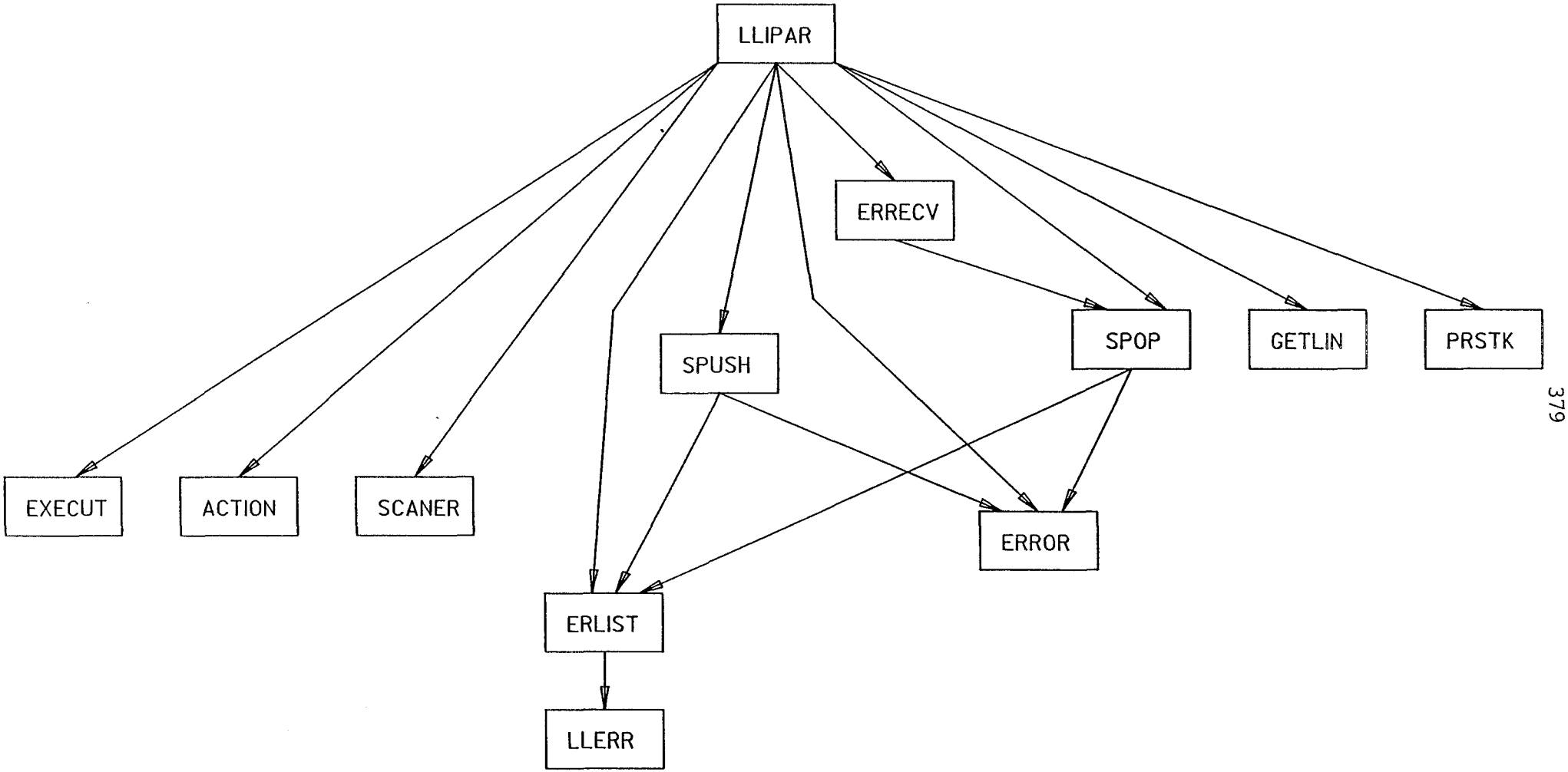


Figure 43. Structure Chart for the Prototype Design System Parser Routines

the parser continued parsing the command stream, executing the code generated when the appropriate token arrived.

The language given in section 1 is not however a LL(1) grammar and must be expanded slightly and then restricted by the semantic portion of the interpreter as demonstrated by Tremblay. This is the type checking of the command stream and is done in the semantic portion of the interpreter. The expanded grammar is given in appendix II.

The parsing function used was that given by Tremblay. The function is best represented by a data table, which also allows for easier modification to the language than coding the parsing function. The LL(1) parsing table for the prototype system is also given in appendix II. During parsing, nonterminals were expanded and pushed onto the parse stack. Because of this, it was necessary to store the right-hand side for each production (or expansion rule). Again the right-hand side productions were implemented as a data structure to facilitate change. The implementation of this aspect is similar to Tremblay's.

Figure 43 illustrates the calling sequence of the parsing procedures. SCANER has already been discussed while ACTION, the context sensitive checker and code generator, is discussed later in this chapter and EXECUT, the run-time interpreter, is discussed in the chapter on the run-time interpreter.

SPUSH pushed the tokens of the right-hand side of the given production onto the top of the parse stack. The SPOP procedure popped the top element off the parse stack. For debugging purposes PRSTK printed out the contents of the parse stack. The error recovery routine, ERRECV, was a much simpler effort than Tremblay's. It had to recover from the error so that the wagon designer could enter his next

command. When a syntax error was detected ERRECV popped tokens off the stack until a token representing the beginning of a session or statement list was found, thereby removing the last command. An action symbol was then pushed onto the stack to reset the code pointer as a consequence any code that might have been generated for the command with the incorrect syntax would be ignored.

The implementation was based on that of Tremblay but with modifications to allow programming in the compatible FORTRAN IV language chosen as the encoding language, and to permit interactive execution of commands one at a time. This command by command execution was done by placing an execution symbol at the end of each command in the expanded grammar. In the LL1PAR routine this symbol was treated as a special action symbol which invoked the run-time interpreter. If the run-time interpreter returned with a flag set indicating the suspension of an instruction, the next token input to the parsing function was the SUSPEND terminal.

(3) Table Handler

The symbol table was used to store useful information about identifiers. Information required was the name of the identifier, the type of the identifier (for example, REAL or STRING), the use of the identifier (for example, simple identifier or method identifier), and the storage location of the value or the instruction identifier. This information was used by the code generation routines to emit the correct code and check semantics and also by the run-time interpreter to link run-time storage addresses with variables stored in the database, and to retrieve instruction identifiers for methods.

The table handling routines were used to perform insertion and lookup operations on the symbol table. Lookup and insertion (if the lookup failed) operations were done by the scanner. A symbol table index was returned from these operations which was subsequently used by the semantic routines. A symbol table initialisation routine was used to insert method, model and variable identifiers at startup time.

As all identifiers were global in scope, the symbol table and table handling routines were simplified versions of those presented by Tremblay. No procedure information table was implemented because procedures were not a feature of the prototype language and information on the solve command parameters was stored in the database. No cross reference table was implemented either.

The symbol table structure employed a hash-table method of indexing, the division method for the hashing function, and separate chaining for collision resolution (Tremblay and Sorenson, 1982).

The table handling routines consisted of three procedures as shown in figure 44. STDLOK read information about system defined identifiers off an initialisation file, performed insertions into the symbol table and assigned storage locations. ERROR and ERLIST were called to print error messages. LOOKUP was invoked to perform search and insert operations for the scanner, while MLOKUP performed run-time lookups. Both LOOKUP and MLOKUP were passed the identifier name and returned the symbol table index of the identifier.

Because they were unavailable in the encoding language, FORTRAN IV, some routines for basic character string handling were written (for example, string comparison, determination of string length, store a string and return the storage location). Strings of dynamic length were

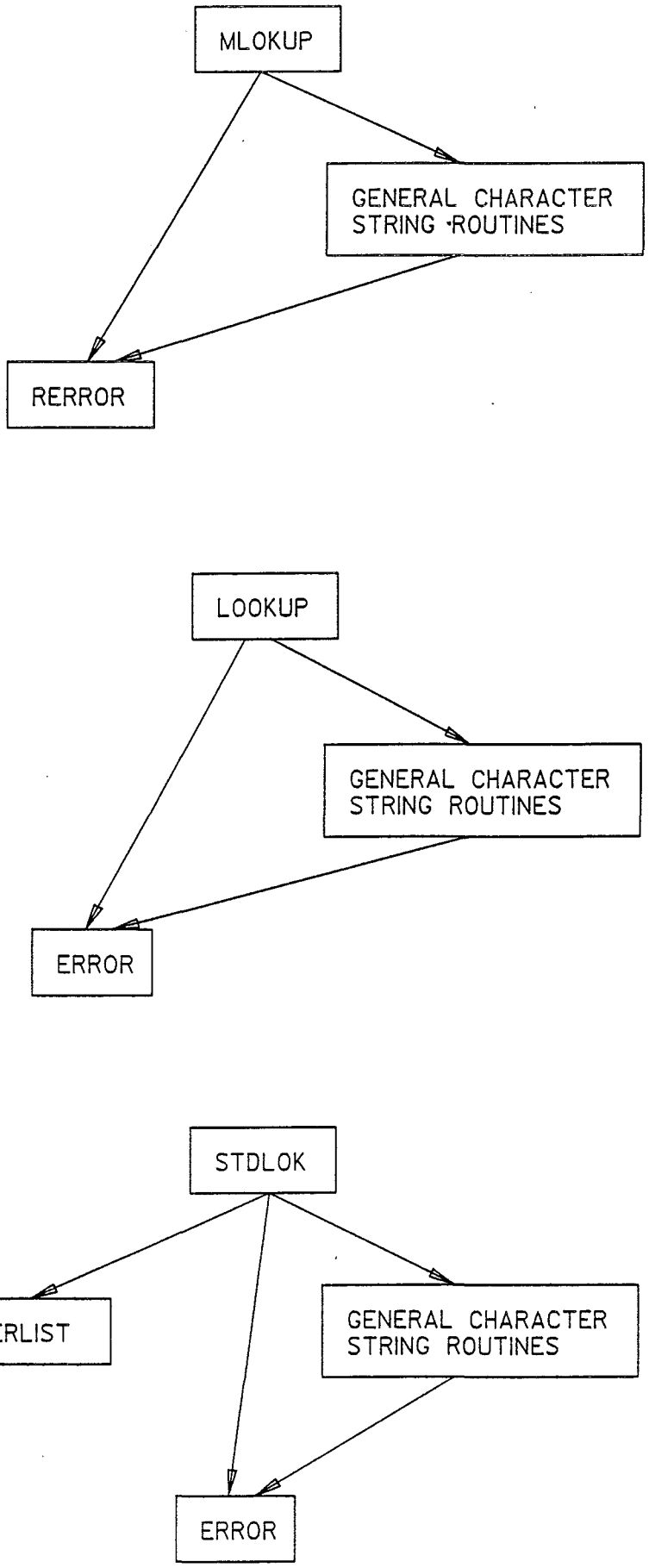


Figure 44. Structure Chart for Table Handler Routines

stored contiguously in an integer vector and two routines were used to find free space for a string and to manage the linked list of free blocks. The name field in the symbol table was a pointer into this string storage vector.

(4) Code Generator

This section introduces the run-time system and then describes the routines used to implement code generation and semantic checking.

In order to implement the code generator the target machine/interpreter must be fully defined. The command interpreter generated intermediate code that was directly executable by a stack oriented run-time interpreter. Although this form of execution is slow in comparison to translating the intermediate code into machine code, decoding overheads are small and a higher degree of portability is achieved. Compute or input/output intensive operations (such as assembly of finite element stiffness matrices) were compiled into machine code.

The run-time interpreter consisted primarily of a code area and a data area. The code area contained the intermediate language instructions. Each instruction was composed of an operation and zero or more operands. These instructions and instruction components were stored contiguously in the code area. The data area consisted of a string storage area, an area for storing variable values and a run-time stack. The sizes of all three data areas were fixed at system compilation time.

The run-time stack was used for passing information to the database accessing routines in the solve command as well as expression

evaluation, assignment, and so forth. The storage requirements for a solve command were organised into a contiguous area called an activation record, similar to that described by Tremblay for executing procedures. The execution of another solve command while one is suspended is analogous to an executing procedure calling another before it has returned. A special location called the activation base pointer contained the base address of the activation record of the currently executing solve command.

All addressing of storage locations in the run-time interpreter was absolute, or in other words, it used the position number in the global storage area. Storage locations for strings contained a pointer (that is, address) to a location in the string area. The first location in the string area contained the string's length. Management of this string area was done by the routines described in section 2(3) of this chapter.

The strategy for implementing code generation and semantic analysis was to augment the LL(1) grammar with symbols called action symbols (but still meet the requirements of a LL(1) grammar). The LL(1) parsing function can be altered to recognise these action symbols and invoke the appropriate routine (Tremblay and Sorenson, 1982). The main advantage in using this approach is that the grammar can be easily changed to incorporate or remove action symbols. When the parser reached an action symbol on the parse stack, following stacking of productions of the grammar according to the parsing function, it invoked the appropriate action routine in order to carry out the semantic analysis and code generation associated with that particular language construction. The augmented grammar for the prototype design system is presented in

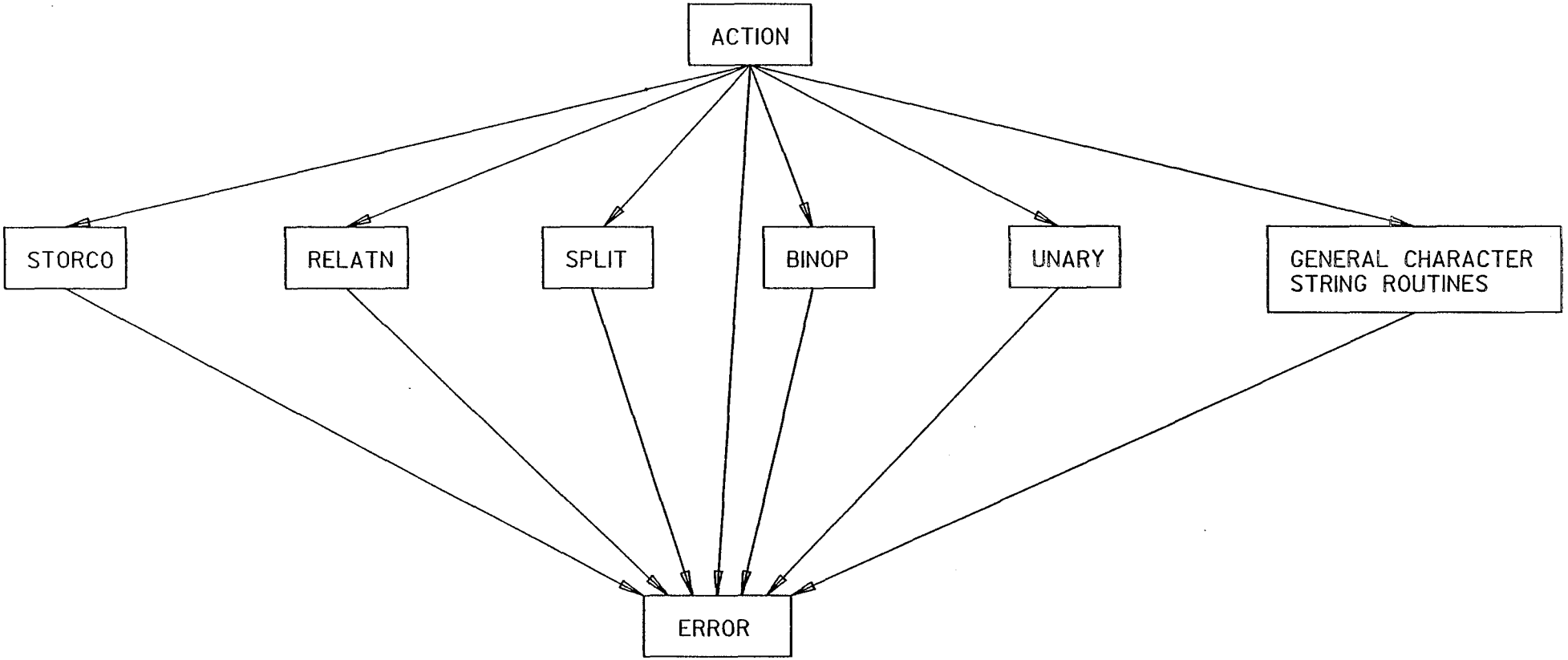


Figure 45. Structure Chart for the Code Generation Routines

appendix II.

The other major component of the command interpreter environment was the semantic stack. The semantic stack was used to store references to other data structures such as the symbol table or other elements in the stack, to store data and variable types, variables controlling the state of the interpreter, and other information on the processed part of the session. Part of its function was to simulate the evaluation of the execution.

The code generator routine ACTION invoked a number of subordinate procedures as illustrated in figure 45. BINOP, UNARY and RELATN performed semantic checking and code generation for binary, unary and relation operations respectively. They were FORTRAN IV implementations of Tremblay's routines with the exception of the restrictions on string operations mentioned earlier. SPLIT tested the type of the first operand for logical conjunction and disjunction and then generated the appropriate code. Again this routine was a FORTRAN IV implementation of Tremblay's. Because the code area and the semantic stack were implemented as integer vectors, real constants were stored temporarily in a real array till execution time. STORCO stored the real constant in this array, returning its address. The type and composition rules implemented by these routines are presented in appendix III.

The next section describes how the various language constructs were handled. A description of the action routines is given in appendix IV.

(a) Primaries. Code was generated to place the value of the primary (or the pointer to the string) on top of the run-time stack. The type of the primary was placed on top of the semantic stack for subsequent type checking once the use of the primary was known. The

action routine for handling real primaries called STORCO, and emitted the code to retrieve the real value from the temporary storage and place it on the run-time stack. The routine for simple identifiers did not allow new identifiers in expressions as their values would be undefined.

(b) Operations. Following Tremblay, the intermediate language was designed so that all operands were placed on top of the run-time stack prior to execution of arithmetic, string, or logical operations. Following execution the operands were replaced with the result of the operation. The types of the operands were placed on top of the semantic stack, and once checked they were removed and the type yielded by the operation was placed on the semantic stack.

A number of extra unary action routines were included to handle the built-in functions, and valid string operations were reduced to equality and inequality.

(c) Assignment. The action taken at the beginning of an assignment command pushed the identifier index (into the symbol table) onto the semantic stack. After processing the right-hand side expression, if the type of the identifier matched that on the top of the semantic stack then the code, to store the value on top of the run-time stack at the identifier's address, was emitted. The store instruction emitted depended on whether the identifier was of a string type or not. If the identifier was a new identifier then it was given the type on top of the semantic stack and the address of a free storage location was saved in the symbol table.

(d) Control and Output Commands. The stop and QL action routines generated the respective instructions to perform these commands. The routine handling the return command emitted the return instruction and

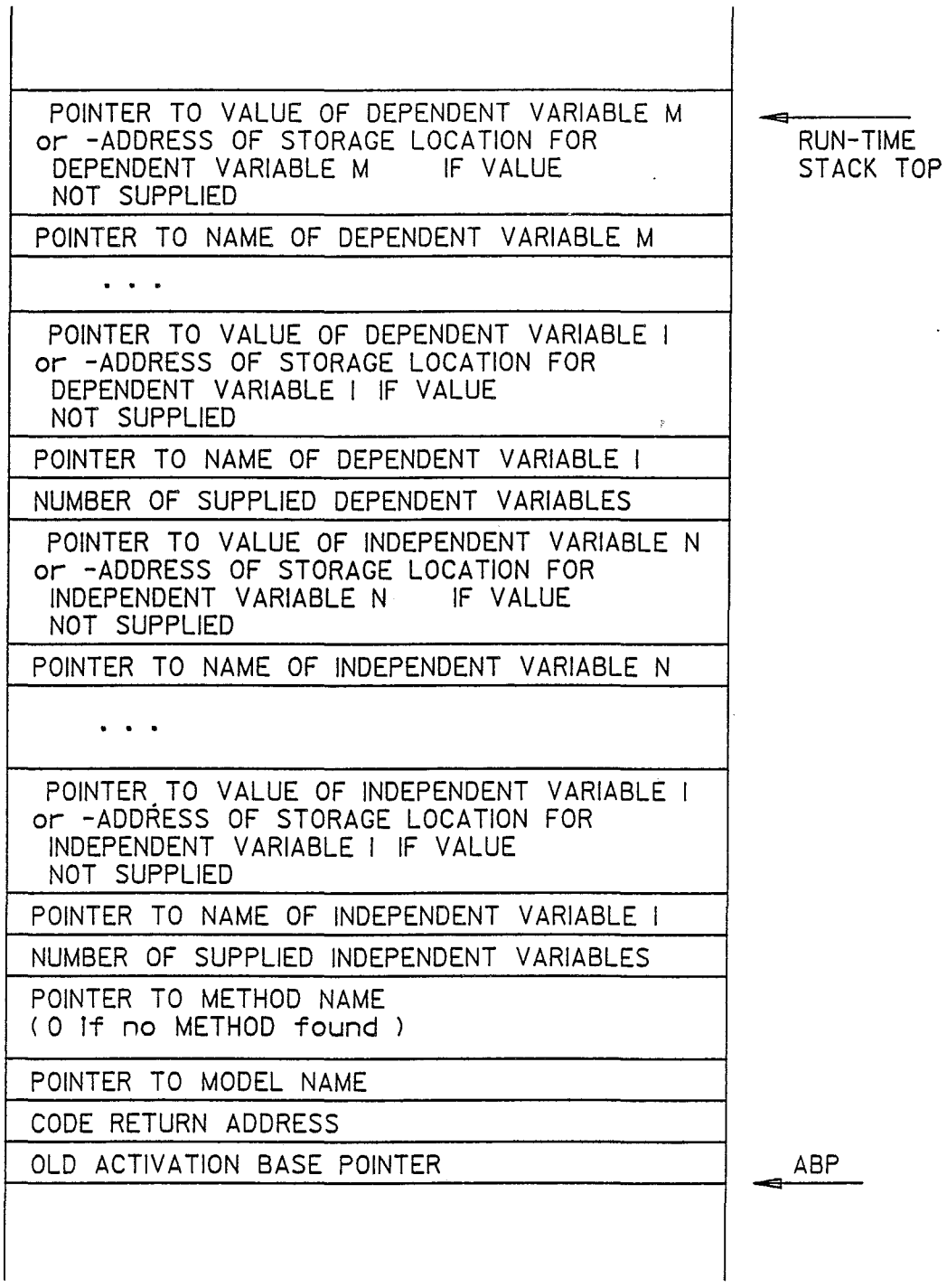


Figure 46. A Segment of the Activation Record for a Solve Command

set the code pointer (the location of the current instruction in the code vector) so as to execute the return instruction next.

At the beginning of the write statement an action routine generated the write initialisation instruction, which had one operand, the output unit identifier. When an expression had been processed an action routine generated the code to place the value in the output buffer according to the resultant type of the expression. At the end of the write command the code to output the buffer to the unit was generated.

(e) Solve Command. The code generated from the solve command constructed the segment of the activation record, as illustrated in figure 46, checked that the independent/dependent variables were defined in the database, that the dependent variables were valid solutions using the model and method, checked if any other dependents were to be calculated, and finally generated the code to retrieve the database data and check for the presence of the dependent variable(s). The activation record was used to communicate model, method and variable data to the run-time interpreter.

The first action routine in the solve command generated the code to save the activation base pointer and the return address in the code array. These intermediate language instructions copied the activation base pointer on top of the run-time stack, and pushed the pointer to the current instruction on to the top of the stack.

The second action routine encountered generated the code to push the pointer to the model name (stored in the string area) on to the top of the run-time stack. The next action placed a set marker on the semantic stack indicating the beginning of the set of dependent variables.

Both the independent and dependent variables were handled in the same manner in the solve command. If a literal was detected, it was stored and the pointer to it placed on top of the semantic stack, and the symbol table index for the variable identifier was then placed on the semantic stack. If no literal was present, a value indicating "no literal found" followed by the symbol table index for the variable identifier were placed on top of the semantic stack. These actions were repeated until the end of the dependent or independent variables was reached, at which point another set marker was placed on the stack (refer fig. 47).

If no method was specified action routine 209 placed a value on the top of the semantic stack indicating "no method found" and generated the code to place a zero value on top of the run-time stack. The action routine handling a method specification was the same as that used for the model specification.

At the end of the command, routine 208 was run if any independent variables were detected. A flag was set indicating that the default method should be sought if the top of the semantic stack contained the "no method found" value. Using the symbol table index, code was generated to place the pointer to the independent variable(s) name(s) on top of the run-time stack. If the next value in the semantic stack was the "no literal found" value, the code to push the negative of the variable address on the run-time stack was emitted. Otherwise code was emitted to assign the literal to the variable (checking beforehand that the variable was of string type) and to place the pointer to the literal on top of the run-time stack. Processing continued until the set marker was detected, and then the space in the code reserved to place the

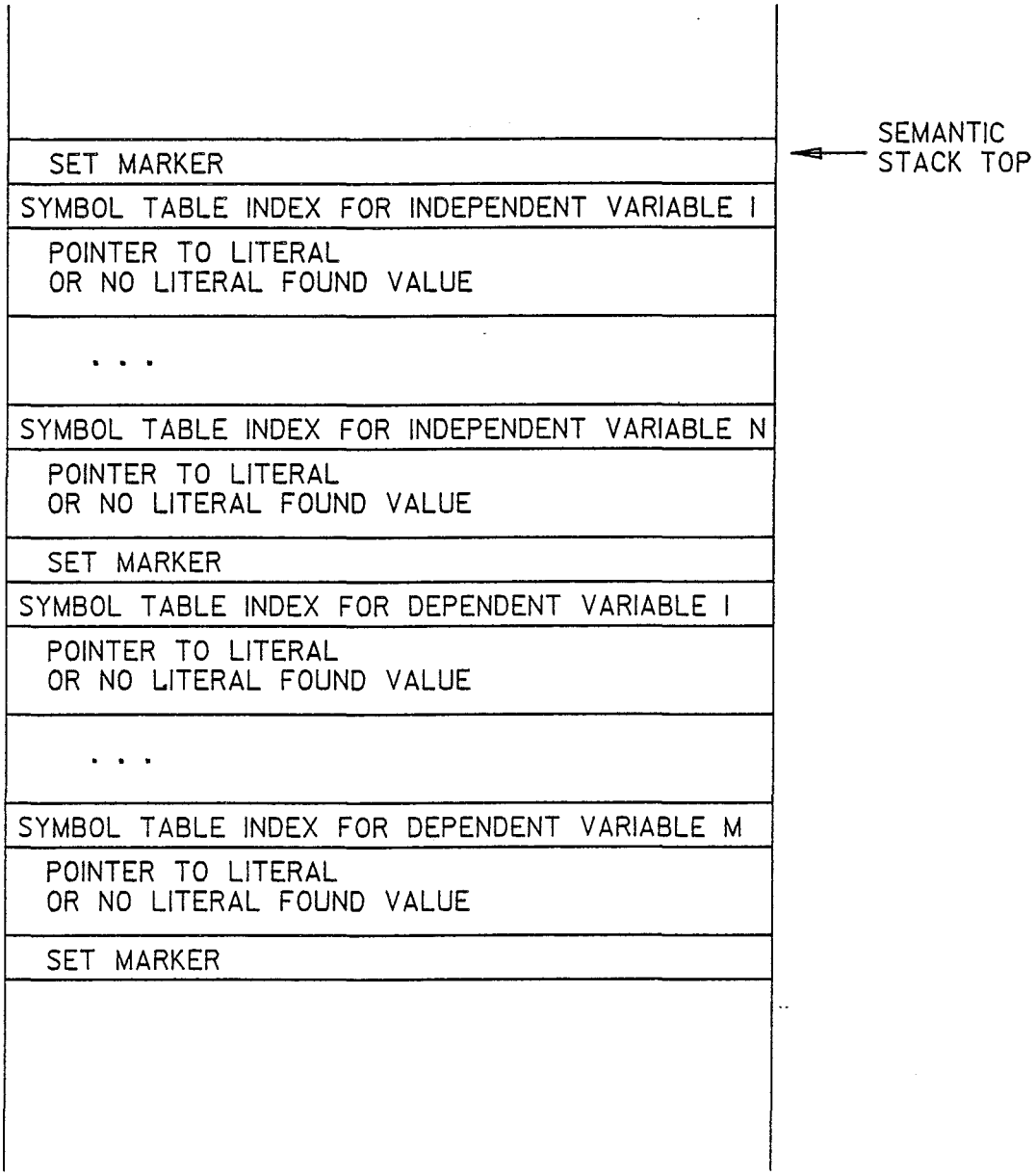


Figure 47. Semantic Stack Following Processing of Dependent and Independent Variables

number of independent variables supplied on top of the run-time stack was filled in. The dependent variable(s) are transferred from the semantic stack in the same manner. Finally code was emitted to reset the activation base pointer, get the default method if one was not supplied and check the validity of model, method and variable entities.

If no independent variable(s) were supplied, action routine 207 was run which emitted code to place zero (as the number of supplied independent variables) on top of the run-time stack. Otherwise it performed the same actions routine above.

(f) Suspended Solve Command. The code generated following the suspension of a solve command could return data to the command, quit the execution of the command, or enable execution of other commands with the execution of the original solve command still suspended. The return command permitted resumption of a previously suspended solve command.

The first action routine encountered when returning data or initiating new commands was that which generated code to start a new activation record. The save activation base pointer intermediate instruction was emitted, as was code to push the current instruction counter onto the top of the run-time stack. Code was generated to reset the activation base pointer and the instruction counter was reset to the save activation base pointer instruction. Action routine 248 was then run to begin a new command.

If data was to be returned to the suspended solve command, action routines 249, 252, 253, and 254 emitted the appropriate code. Routine 252 generated code to push an integer value onto the run-time stack and then copy it to one below the activation base. When a literal was entered, routine 253 generated the code to store the string, push the

string pointer on to the top of the run-time stack and then copy it to one below the activation base. If the default was accepted, code was emitted by routine 254 to copy the value at one below the activation base on to the top of the stack, replace that with the absolute value and then load it back into one below the activation base. When an identifier name was to be returned to the suspended solve, the action routine 249 generated code to push the string pointer recovered from the symbol table on to the top of the run-time stack and then copy it to one below the activation base.

Following the processing of the data, code was generated to return to the suspended solve command. This code made the old instruction counter and activation base pointers (the ones stored at the beginning of the activation record) the current ones. The top of the stack was set to the old activation base pointer minus one. This routine was not run if a new command had been entered.

The quit command initiated the new command action routine and then generated the code to perform a return or unstacking of a solve command and its activation record. This code reset the activation base pointer and instruction counter to those stored in the activation record, set the top of the run-time to activation base pointer minus one (as above), but then, if it was returning to another suspended solve command, output a message informing the wagon designer about the suspended problem. The instruction counter was reset to the beginning of this new command. The return command used the same action routine to generate code. The intermediate language return instruction checked that a return or unstacking to beyond the primary level was not being attempted.

(g) Immediate Execution. At the end of all commands action routine 243 was run to generate a stop instruction. The next action symbol normally encountered invoked the run-time interpreter.

(h) Error Recovery. When the parser recovered from a syntactic error (by invoking ERRECV, or actioned in GETFP which although called from SCANER performed syntactic analysis of real numbers) then a flag was set indicating no code was to be generated. This flag could also be set by ACTION and its subordinate procedures. If this flag was set on entry to ACTION then all semantic processing was ignored until an action symbol signalling the start of a new command was encountered. When this occurred, the semantic stack and code area were restored to the state they were in prior to the processing of the command containing the error.

3. COMMAND INTERPRETER RESPONSES

Interpreter responses consisted of error messages. Appendix V provides a listing of the errors that could arise in the scanner, table handler, parser or code generation routines. All errors were reported using the ERROR and ERLIST procedures.

The arguments passed to ERROR included the error message number and parameters for the error message. The effect of the ERROR procedure was to insert the error message number and parameters in an error buffer. In addition, it marked the location within the line of the error. ERLIST displayed all the error messages in the buffer, calling LLERR to generate messages for parsing errors.

4. REFERENCES

DAY, A.C. (1972) Fortran Techniques. C.U.P.

TREMBLAY, J.B. and SORENSON, P.G. (1982) An Implementation Guide to Compiler Writing. New York, McGraw-Hill. 259p.

CHAPTER XVII

DESIGN AND IMPLEMENTATION OF THE PROTOTYPE DATABASE

This chapter describes the detail design and implementation of the database for the prototype wagon design system. The methodology was loosely based on Buchmann's (1981) SIMBAD.

An implementable subset of data elements have been studied to demonstrate the feasibility, utility and convenience of the proposed scheme, and repeated application of the same approach to other subtasks will result in a more comprehensive global view. Development, particularly that expanding the global view, should be regarded as normal.

1. GLOBAL ARCHITECTURE

Analysis at a global level of the data handled in wagon design showed three underlying categories of data. Approved project data are those having gone through the approval process and are ready for dissemination to others working on the project. Catalogue data includes vendor information on components, national and international standards, article and paper abstracts, and so forth. The last category is temporary data. This data is changing rapidly as it is created and modified, and it is not ready for submission for approval and dissemination. Figure 48 displays the characteristics of the categories of data.

	Relative Acceptable Response Time	Relative Volume	Relative Rate of Updates	Relative Rate of Accesses	Relative Number of Users
Approved Project Data					
- Old	Medium	Monotonic Growing	Low	Low	High
- Current	Small	Monotonic Growing	Low	Medium	High
Temporary Project Data					
- Checkable	Small	Variable	Medium	High	Medium
- Personal	Small	Variable	High	High	Low
Catalogue Data	Medium	Low	Very Low	Medium	Medium

Figure 48. Characteristics of the Design Data Catagories

Approved data undergoes a low rate of revision as problems are found and fixed but is accessed by a large number of people. Approved project data steadily accumulates as projects reach completion. Temporary data could be divided into categories according to the number of users accessing it, because the wagon designer submits his work for checking by others and it is also used as a tentative solution referenced by other wagon designers. The checkable sub-category of temporary data in figure 48 illustrates this.

These differing characteristics suggest breaking up the database into at least three parts: approved project database; a set of smaller databases that serve as workspaces and where the wagon designer can save preliminary alternatives; and the almost static catalogue database.

Attempts were made to design aspects of all three databases although more effort was expended on the temporary data as it is at least as wide in scope as the others and is the one with which the wagon designer has the most interaction.

2. DEFINITION OF DESIGN DATA

In this step the data objects are defined together with processing, distribution, access control, archiving requirements and consistency constraints. Homonyms and synonyms are resolved.

(1) Catalogue Data

The catalogue data studied were the physical properties such as density and Young's Modulus of materials and the geometry, section, and mass properties of four British Standard rolled sections, all of which

are available in manufacturers' catalogues and handbooks. The data definitions can be found in appendix VI for these items. All users should be notified of any changes to these data elements.

The local views that have been modelled include:

(i) interactive queries and application retrievals of one or more dimensions/properties given the material name(s)/section nominal size(s) and possibly thickness(es) or mass(es) per unit length;

(ii) section nominal size(s) (or other attributes)/material name(s) (or other attributes) of sections/materials meeting specified selection criteria, for example, UTS > 350 MN/m^2, or 350 < Yield stress < 450 MN/m^2 (note the selection criteria could be a combination of criteria on many attributes).

(2) Approved Project Data

The source of approved project data used in this part of the study were the Design Office approved drawings. Drawings form the bulk of approved project data, other sources include manufacturing instructions in the form of letters or Loco 204's and in recent years manufacturing specifications.

Approved drawings may be a list of parts, wagon "diagrams" (overall dimensions, weight, major assembly drawing number reference), wagon load limits, schedule of tolerances, weld symbols, working drawings, and so forth. The geometry that could be parameterised and the non-graphical data recorded about parts, assemblies, and wagons were studied in detail.

The local views allow for the retrieval (and creation) of this information for the purposes of analysis (for example, the number of

coils, wire diameter, and so forth, for a coil spring) and the retrieval of the geometry of a part or assembly to assist part rationalisation. These are high level views that have more meaning than a list of displayable lines and curves. However, drawings and/or graphical representations as output would be possible by using the parameters of these high level descriptions in algorithms designed to produce the required graphics. Creation using graphical display techniques is also possible by using similar algorithms. Adding features (and their sizes and positions) such as a "throughhole" and "chamfer" to a "flat plate" or an "end shape" to an "equal angle rolled steel section" is possibly a more natural way to design than placing lines and surfaces.

Storing the geometrical data in parameterised form would appear to incur a performance penalty for display purposes but would be more storage efficient than lower level display list type storage methods. Approved data is not updated as frequently as the working temporary data and therefore performance penalties incurred with a parameterised feature approach are less important. Information is, however, directly available for analysis and does not require human interpretation as occurs when the data is only stored as lines and curves. Since the design is stored in terms of features, overall shape, and their parameters, a more automated approach may be possible for the manufacturing operations and for relating production information (capabilities and costs) to the wagon designer.

There is however a cost in preparing and defining the features and parameters, which may not be always justified compared to more general approaches that store geometry in terms of lines, curves, surfaces or primitive solid instances.

The data definitions can be found in appendix VI.

All users who are currently using the approved project data or who have taken copies of any of the data should be notified of any changes. Mailing lists can be drawn up for the release of each new set of approved data.

(3) Temporary Project Data

The temporary project data modelled included that comprising a local view of the author's finite element program. The printouts from a number of the Design Office's more recent STRUDL runs were analysed to determine how in practical usage, data elements were related.

The complete data model of this local view was not finished - only enough was completed to test the first two modules of the finite element program, namely, the formation of the stiffness matrix (including degree of freedom numbering), and the formation of the right-hand side matrix. With more computational models included, more entities would have MODEL and DATE attributes.

The second local view of temporary project data was that of modelling and methods. Part of this view was a data dictionary containing data elements, data groupings, and so forth. The other part contained specification of valid relationships between variables (that is, the models), valid solution definitions, default solution methods, and other pertinent details on variables and models.

The data element definitions for both views can be found in appendix VI.

Generic definitions of attributes for models and variables are included in the models and methods' local view. Two methods, STIFFEN

and ASSEMBLE RHS are listed and their variables may be found in the finite element analysis local view data definitions.

Any user who has been granted access to another user's temporary project data should be notified of any changes.

3. LOCAL VIEW DATA ELEMENT DEPENDENCIES

In this section the associations between the data elements in each local view are presented.

(1) Catalogue Data

The associations between the data elements for the catalogue data are contained in appendix VII. Dimensional/material property data is identified by one or two data elements and sectional properties are dependent on dimensions and shape.

(2) Approved Project Data

The dependencies between approved project data elements are in appendix VII. Each engineering change order (Loco 204 or drawing amendment) changes the revision of parts and assemblies. There are also versions of a class of wagon. The attributes functionally dependent on the combination of version or revision and item identifier or wagon class are fully dependent on this key and are not just dependent on wagon class or item identifier. Therefore the original designer of a product is the designer of version A.

Load limits are more involved than presented here as there can be limits for any number of loading patterns, but for the prototype scheme

this was ignored.

Most assemblies and piece parts refer to the wagon correspondence file, but there are exceptions, mainly for common components such as load plates and bogies.

RETRIEVAL CLASSIFICATION, an Opitz shape classification number, is fully functionally dependent on both PART NUMBER and REVISION because revisions can be of such a magnitude that a dimension change could put the part into another retrieval class. RETRIEVAL CLASSIFICATION acts as a coarse indexing system into part geometries. Opitz classification was not used by the Design Office but it is the author's opinion that it offers many benefits even in a computerised environment, and has therefore been introduced into the prototype scheme.

The PART NUMBER and ASSEMBLY NUMBER are usually the drawing number reference, or for bought-in items, the Stores Branch stock item number, the manufacturer's drawing number, or part number. The stock item numbers and drawing reference are such that there is no overlap.

The PIECE PART attribute STANDARDNESS indicates whether the part is in the Design Office's standard part library. The MATERIAL IDENTITY attribute relates the PIECE PART with MATERIAL PROPERTIES. The GEOMETRICAL ASSEMBLY TYPE and GEOMETRICAL ASSEMBLY NUMBER are the relation and tuple in that relation, respectively, where the piece part's geometry is described. Therefore all relations that can be referenced in the PIECE PART relation must have a single attribute primary key defined on the GEOMETRICAL ASSEMBLY NUMBER domain.

In the relationships that assemble items, INSTANCE is used to identify multiple occurrences in the global axis system of the same item. The inclusion of revision levels for the assembly, sub-item and

wagon permits the automatic creation of a history of changes to the relationships. They allow revision levels of piece parts to be changed independent of revision level of the assembly and they also allow the assembling details of different revision levels to exist at the same time, which is useful in a world in which there are delays in receiving and manufacturing parts. Tuples must be inserted when either the assembly or sub-item revision level changes, thereby making visible the effect of such changes.

The positioning of parts and assemblies involves translation and rotation but no scaling. Translation in each of the global axes and six direction cosines are stored for each part in an assembly. These values were stored as opposed to Eulerian angles, quaternion elements, co-ordinates, or other forms because it was thought they offered the best compromise between storage efficiency, calculation efficiency and naturalness to the wagon designer. There must be one consistent set of global axes for each ASSEMBLY, and a datum and axis for each sub-item.

These general assembling relationships could be replaced in part by more specific assembling relationships for common assemblies, for example drawgear in wagons, bushes in brake rods.

It should be noted that from these relationships (since they contain more complexity than a normal product structure), part list reports, bill of materials reports, and, where used, reports can be obtained.

The GEOMETRICAL ASSEMBLY NUMBER or primary keys defined on the domain of the same name were introduced to provide an identification attribute. Another candidate key is possibly all other attributes in the geometrical relations except this attribute. But uniqueness for all the attributes except the GEOMETRICAL ASSEMBLY NUMBER in an updating, inserting, deleting environment may not be desirable. These primary keys are defined on the same GEOMETRICAL ASSEMBLY NUMBER domain because each is a foreign key in PIECE PART and GEOMETRIC FEATURE ASSEMBLY relations. GEOMETRICAL ASSEMBLY NUMBER and the other primary keys must be unique across all the geometrical relations where the primary key is defined on the GEOMETRICAL ASSEMBLY NUMBER domain.

Each geometry definition consists of either an overall shape (or type) with attributes such as thickness or length and a number of features or further details that make up the complete definition or else the definition consists of an assembly of geometrical features. For example, a tie rod consists of round bar with rod eyes, cylindrical end or threaded ends; a beam consists of rolled section with two ends of any shape (and possibly a number of through holes); a flat plate can have any outer shape; a pin may be of black or turned type. Features are assembled in much the same way as assemblies of piece parts. These features allow association of attributes other than just geometry. For example, a rod eye has the attribute of being a connection joint with one rotational degree of freedom. Whether a certain geometry is standard or not is another nongeometrical attribute.

(3) Temporary Project Data

The dependencies for the local views of the temporary project data

are contained in appendix VII.

The associations between entities such as forces and nodal forces, accelerations and body loads, positions and nodes, elements and structures, beam properties and beam elements, and so on, are of a many to one nature. It was found in the STRUDL printouts that they were used repeatedly and it was expected that with more computational models this level of breakdown of entities would be required. For example, container twistlock point loads appeared again and again in load cases, and the same beam properties were used many times both in runs on the same wagon and on different wagons.

Names and identities were added to most entities. Names do not appear for many entities in the existing system but their introduction gives the wagon designer a means of identification. However this attribute is not a candidate key because it was felt unnecessary to enforce uniqueness on this attribute. For the wagon designer the relationships with other data, date of creation, and so forth, would provide unique identification. Therefore the attribute IDENTITY becomes the primary key but it is not intended that the wagon designer use this attribute frequently.

Sets or grouping of entities is another feature of the temporary project database, for example, element groups, groups of load cases, groups of displacements. Some of these groupings are necessary for the finite element programs, but there is also the requirement to be able to identify and manipulate sets of entities.

The generic variable and generic model dependencies are included in appendix VII, as well as the dependencies in the STIFFEN and ASSEMBLE RHS models. Models and variables were made independent of their

respective table names. The variable identifying number was introduced to reduce storage requirements, increase search speeds and provide some independence from the run-time interpreter variable name.

4. AGGREGATION OF DATA ELEMENTS FOR LOCAL VIEWS

Based on the known dependencies, the aggregation step forms the information entities and relationships and identifies the primary and alternate keys. As far as the dependencies identified are complete, the relations will be in 4NF except where practical considerations dictate otherwise. Alternate keys are specified:

(i) to inform the DBMS that corresponding to each alternate key instance there exists a unique primary key instance, and the converse also applies;

(ii) to inform the users, since the uniqueness of the key attributes is an aspect of the semantics of the relation and of interest to people using it.

Alternate keys are then candidates for secondary indexing to improve accessing performance.

(1) Catalogue Data

From the dependencies identified in section 3(1) nominal size or nominal size in combination with mass/unit length or flange thickness is the only candidate key for the section data. But because other local views have entities which reference these sections and the relational model is the implementation model, use of this candidate key as primary key is undesirable. A new candidate key attribute called cross section

number was added to all section attributes. Using this attribute as the primary key gave advantages of smaller storage requirements in other relations that have CROSS SECTION NUMBER as a foreign key. Then cross section becomes an abstract type consisting of cross section shape identifier and particular section identifier (or number in this case). Thus all references to cross sections can be treated in the same manner.

Section properties such as section modulus and area are dependent on shape and dimensions, but because these are standard sections updating is at a very low rate (if at all). Better performance may be achieved by storing these properties rather than calculating them at every access. Also aggregating them into the section entities forms a more "natural" user view. Doing this leaves the section entities in 2NF as the section properties have only transitive dependence on CROSS SECTION NUMBER. Updating the dimensions without updating properties will leave the properties inconsistent with the dimensions. The attribute MATERIAL IDENTITY was added to the MATERIAL PROPERTIES entity to reduce storage size as a material identifier will be a foreign key in entities for other applications.

Appendix VIII contains the aggregated relations.

(2) Approved Project Data

Identifying numbers (defined on the common domain of ITEM IDENTIFIER) are uniquely applied to both assemblies and piece parts. Thus the correspondence file, section number, manufacturing specification, and assembly sub-items relationships can be combined for piece parts and assemblies.

In order to reduce storage requirements the MANUFACTURING

SPECIFICATION attribute has been changed to MANUFACTURING SPECIFICATION IDENTITY and a MANUFACTURING SPECIFICATION relation has been added. This removes the text from the primary key definitions in the other relationships.

The aggregated relations are presented in appendix VIII.

(3) Temporary Project Data

The aggregated relations are contained in appendix VIII. The relations can be formed directly from the data element dependency diagrams.

5. TESTING OF LOCAL VIEWS

(1) Catalogue Data

The catalogue data can be tested in the MIMER/Query Language by commands such as:

```
GET MATERIAL.YIELD WHERE MATERIAL.NAME EQ 'CAST IRON';
GET BSCHAN.SIZE WHERE UB71BS4.SIZE EQ '533*210'
      AND UB71BS4.MPERL EQ '122'
      AND BSCHAN.AREA GT UB71BS4.AREA;
```

The first command retrieves the yield stress of the material named cast iron. The second retrieves the nominal size of channel sections whose area is greater than that of the universal beam 533 x 210 nominal size and 122 kg/m.

(2) Approved Project Data

Data for a number of wagons, assemblies, and piece parts were

loaded into the MIMER database using relations based on the aggregates defined in appendix VIII. The scheme was tested using the MIMER/Query Language to produce part lists and bill of materials for assemblies; retrieve parts by using the retrieval classification number, and section number; report on part geometry, approval dates, and so forth.

The author believes that from this database design the geometry, title block and drawing notes could be reproduced for working drawings by writing the appropriate programs, and that the wagon diagrams and other pictorial representations could also be produced by programming. However dimensioning and tolerancing were areas left relatively untouched by this scheme.

Welding was another area in which the scheme was deficient. A special type of part could be defined with attributes of size, type, and so on, so that length, position, and orientation were specified when assembled with piece parts. Alternatively welds could be described as one type of connection relation with constituent parts' surfaces touching and parts fixed in relation to one another (as opposed to connection relations with rotational and/or translational degrees of freedom).

Also missing were relationships to cater for drawing notes associated with locations or features on a part not just the part itself.

An intermediate or dummy assembly would have been useful for grouping frequently used arrangements of parts, for example, liners on brake beams, and the spring block assembly in the draftgear assembly. This feature would allow translation and tolerancing to be a closer representation of drawing dimensioning and tolerancing.

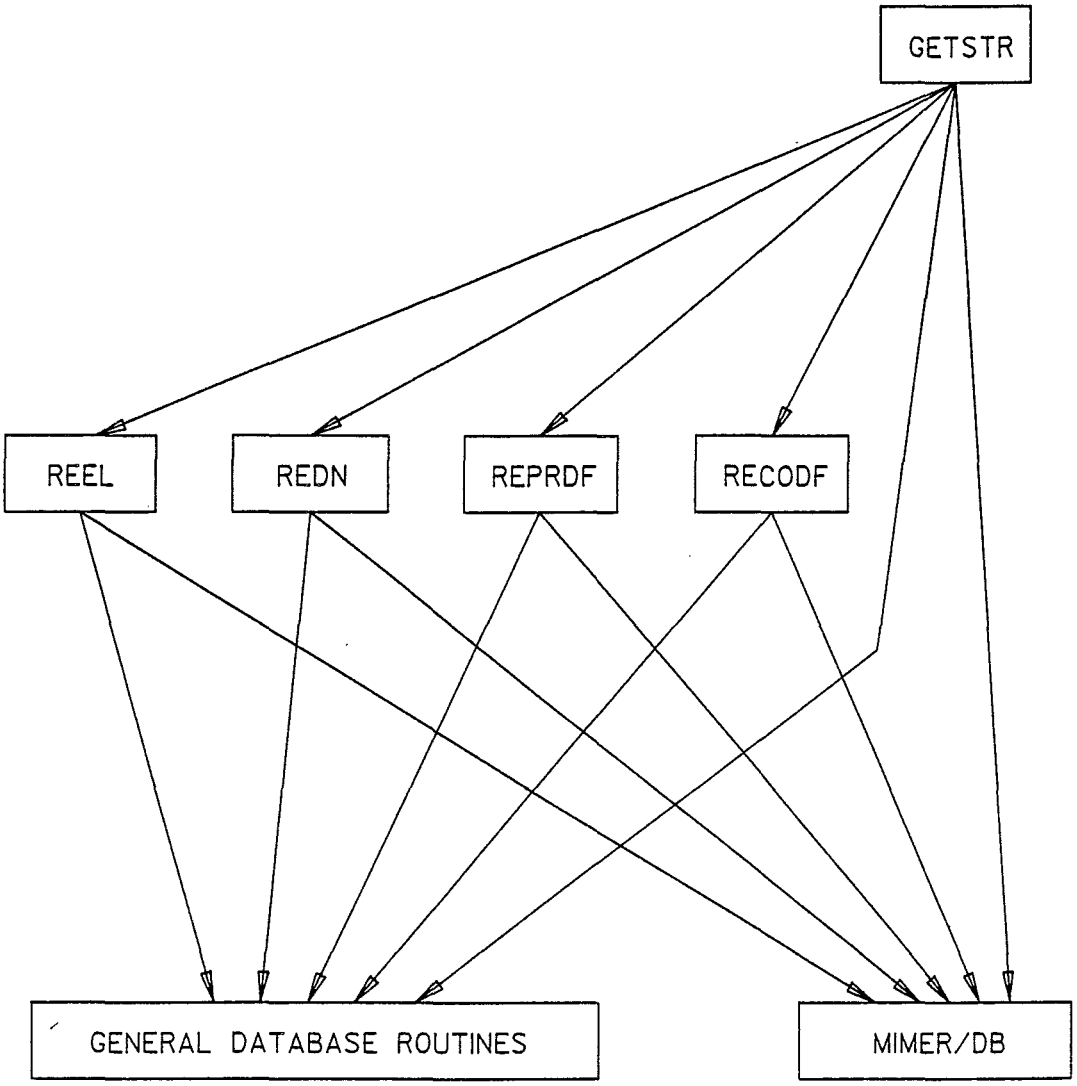


Figure 49. Structure Chart for GETSTR Routines

Translation and rotation matrix data is more difficult to prepare than positioning parts graphically on a workstation display. If performed on a workstation, the part can be positioned by locating existing features on other parts, and then the transformation matrix could be loaded into the database from the display data.

Loading the database with data also showed that some drawings contain geometry only and insufficient geometry to define a part. The missing entity (GEOMETRY ONLY DRAWING) has attributes of approver, revision, drawing number, geometrical assembly number, and so on, but no material, retrieval classification number, or acquisition type.

The GEOMETRICAL ASSEMBLY TYPE attribute in GEOMETRIC FEATURE ASSEMBLY allowed for the general case of many assembly types but was unnecessary for the data loaded.

One way of ensuring uniqueness among the primary keys defined on the GEOMETRICAL ASSEMBLY NUMBER domain would have been to enter the instances in the GEOMETRICAL ASSEMBLY relation. The STANDARDNESS attribute could be removed from the other geometrical relations and placed in this new relation. A GEOMETRICAL ASSEMBLY TYPE attribute could be added to this relation and removed from the others to speed access.

(3) Temporary Project Data

The models and methods local view was tested in the prototype wagon design system (refer chapter XVIII).

Six routines were written to map the logical view to the external view for the testing of the finite element local view of the temporary project data. Briefly they were:

(a) The GETSTR routine retrieved the structure data given the structure identity (refer fig. 49). The procedure REDN retrieved position data from the STRUCTURAL NODE and POSITION relations. In MIMER/Query Language REDN loaded the finite procedure data arrays using the following query:

```
GET POSITION.X, POSITION.Y, POSITION.Z, STRUCTURAL_NODE.NODE_NUMBER
WHERE STRUCTURAL_NODE.STRUCTURE_IDENTITY EQ 'x'
AND POSITION.IDENTITY EQ STRUCTURAL_NODE.POSITION;
```

The REPRDF procedure retrieved data from the PRESCRIBED DOF relation. It loaded data arrays using a query similar to the following:

```
GET PRESCRIBED_DOF.STRUCTURAL_NODE, PRESCRIBED_DOF.DIRECTION
 WHERE PRESCRIBED_DOF.STRUCTURE_IDENTITY EQ 'x';
```

The RECODF procedure retrieved data from the CONSTRAINED DEGREE OF FREEDOM relation. Its query was as follows:

```
GET CONSTRAINED_DEGREE_OF_FREEDOM.STRUCTURAL_NODE,
    CONSTRAINED_DEGREE_OF_FREEDOM.DIRECTION
WHERE CONSTRAINED_DEGREE_OF_FREEDOM.STRUCTURE_IDENTITY EQ 'x';
```

The REEL procedure retrieved the element data. The first part of this procedure performed the following query:

```
GET ELEMENT_IN_STRUCTURE. ELEMENT_NUMBER,
    ELEMENT.TYPE, ELEMENT.TYPE_IDENTITY
WHERE ELEMENT_IN_STRUCTURE.STRUCTURE_IDENTITY EQ 'x'
AND ELEMENT.IDENTITY EQ ELEMENT_IN_STRUCTURE.ELEMENT_IDENTITY;
```

The ELEMENT.TYPE attribute was used to determine what further data was retrieved for each element. For example, for beam elements the following queries were performed:

```
GET   BEAM_ELEMENT.STRUCTURAL_NODE_1,
```

```
        BEAM_ELEMENT.STRUCTURAL_NODE_2
    WHERE BEAM_ELEMENT.IDENTITY EQ ELEMENT.TYPE_IDENTITY;
    GET   POSITION.X, POSITION.Y, POSITION.Z
    WHERE POSITION.IDENTITY EQ BEAM_ELEMENT.THIRD_NODE
    AND   BEAM_ELEMENT.IDENTITY EQ ELEMENT.TYPE_IDENTITY;
```

The uniqueness of rows of data in this last query was assured by loading a "scratch" relation (which had THIRD NODE as the primary key) with the THIRD NODE attribute from the beam elements found and only loading the data array if THIRD NODE was not already in this scratch table. Material data, section property data, and the combination of material data and section properties were also treated in the same way.

```
    GET   MATERIAL_PROPERTIES.YOUNGS_MODULUS,
          MATERIAL_PROPERTIES.POISSONS_RATIO,
          MATERIAL_PROPERITES.DENSITY
    WHERE MATERIAL_PROPERTIES.MATERIAL_IDENTITY
                EQ BEAM_ELEMENT.MATERIAL_IDENTITY
    AND   BEAM_ELEMENT.IDENTITY EQ ELEMENT.TYPE_IDENTITY;
```

The relation and attribute names for section properties depended on the value of X-SECTION_TYPE attribute for the beam element.

```
    GET   x-section_type.AREA, x-section_type. Iyy,
          x-section_type.Ixx, x-section_type.J
    WHERE x-section_type.SECTION_NUMBER EQ
          BEAM_ELEMENT.X-SECTION_NUMBER
    AND   BEAM_ELEMENT.IDENTITY EQ ELEMENT.TYPE_IDENTITY;
```

The general database routines included routines for checking for null values, and handling of errors and database constraint violations. Program and database constraint violation messages are contained in appendix IX.

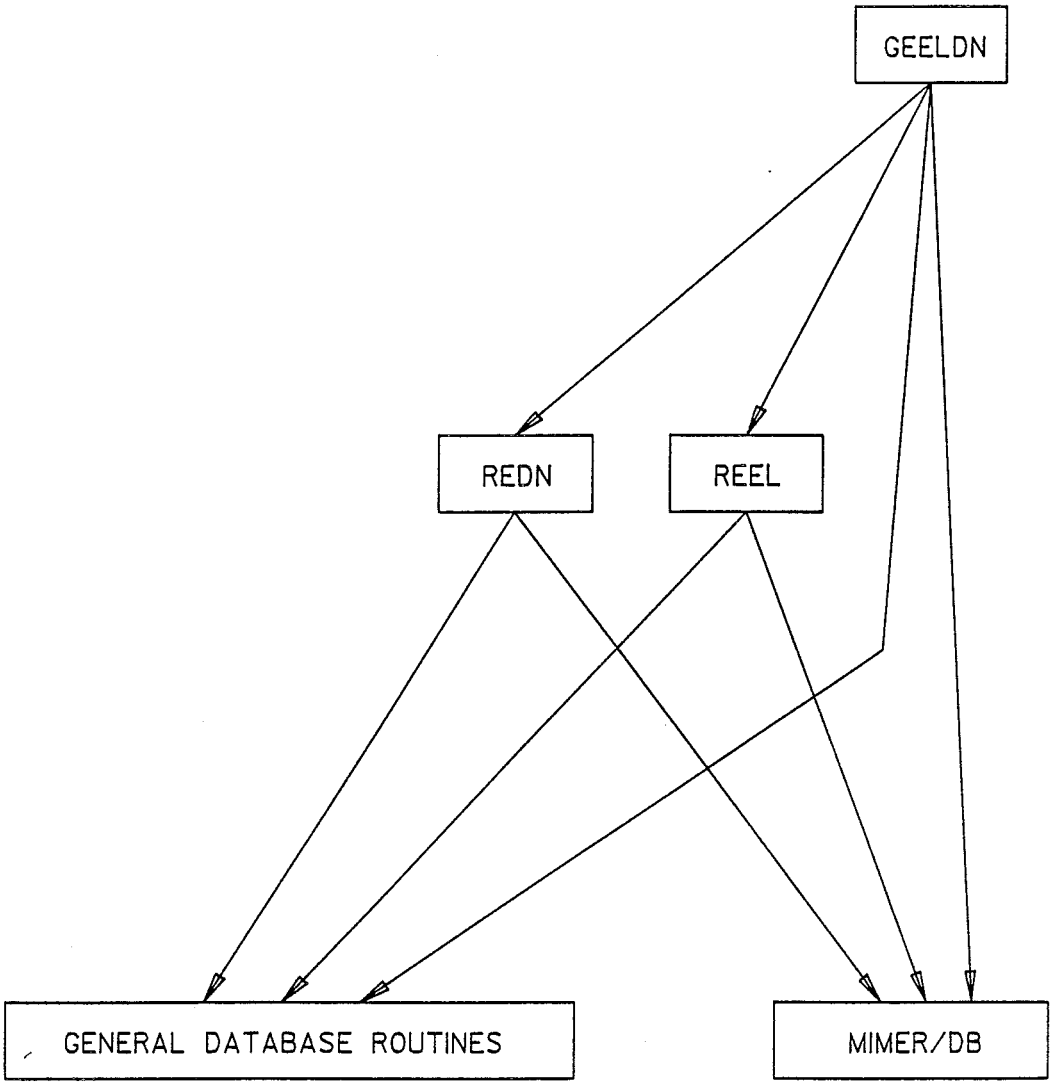


Figure 50. Structure Chart for GEELDN Routines

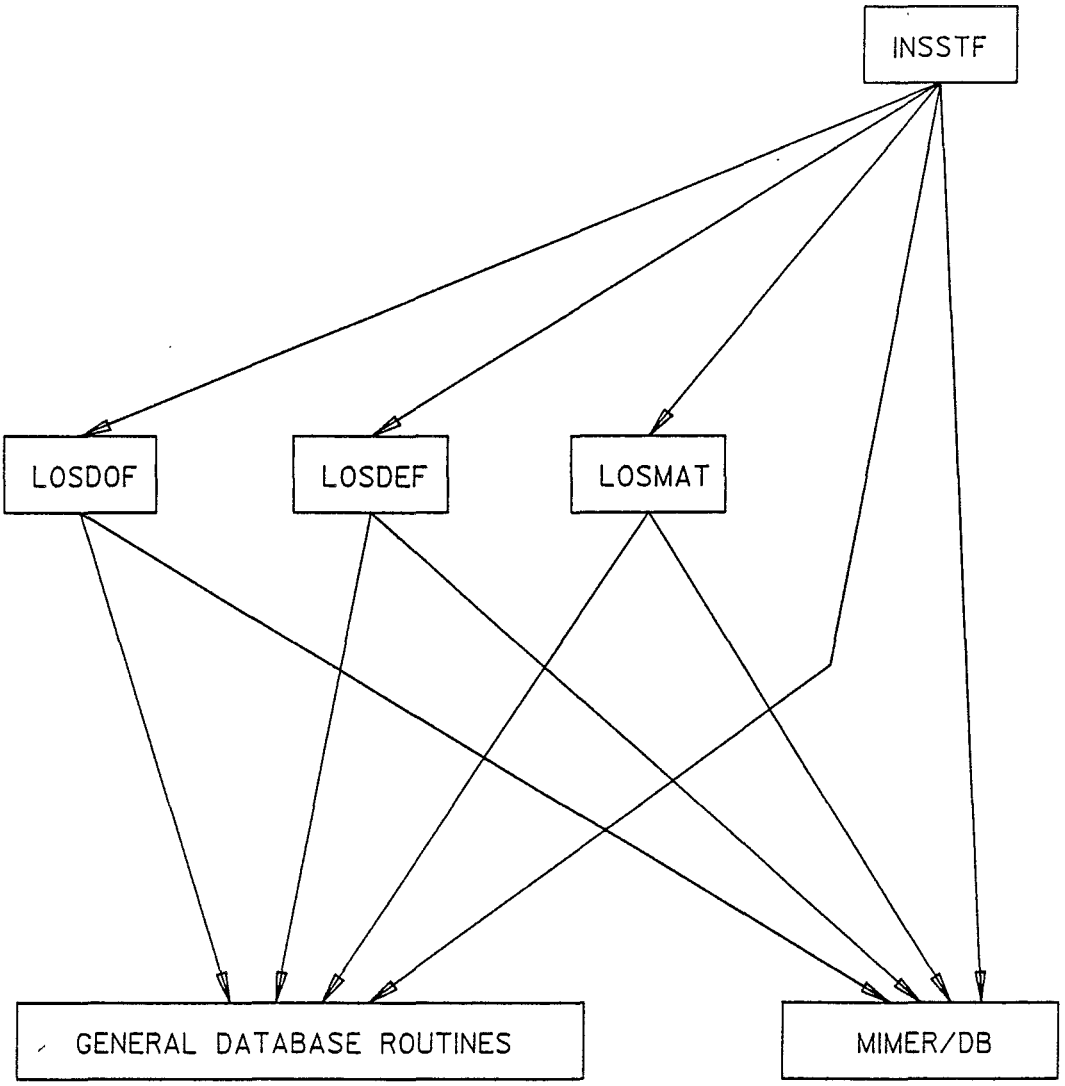


Figure 51. Structure Chart for INSSTF Routines

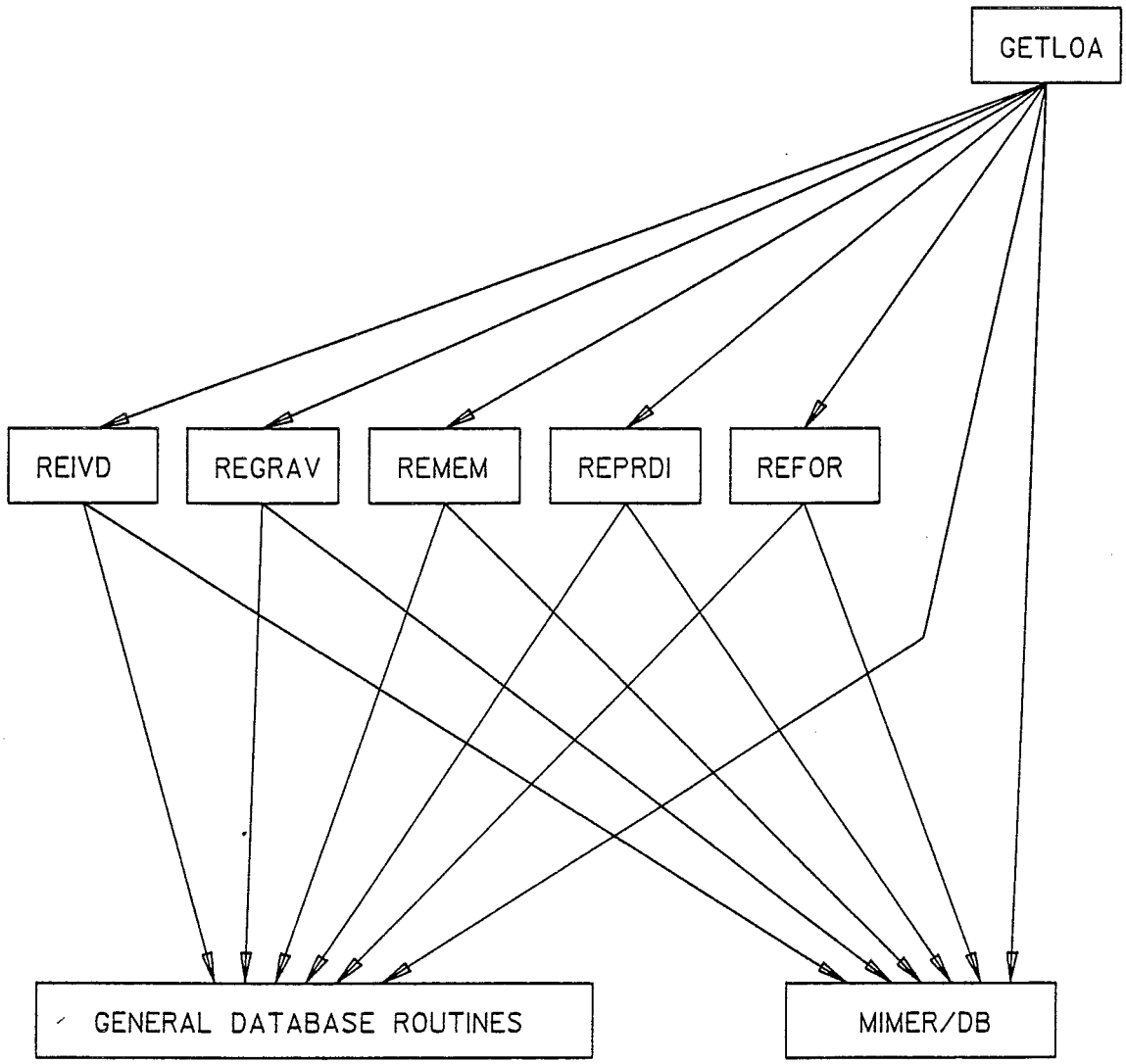


Figure 52. Structure Chart for GETLOA Routines

(b) The GEELDN routine retrieved only element and nodal data. The two routines it called, namely REDN and REEL are described above (refer fig. 50).

(c) The INSSTF routine inserted stiffness matrix data into the database (refer fig 51). It called three routines.

LOSDOF loaded the degree of freedom numbering into the DEGREE OF FREEDOM NUMBERING relation. It required the stiffness matrix identity and inserted a tuple into the relation for each node.

LOSDEF updated the SEMI-BANDWIDTH, NUMBER OF DOF, and TOTAL EQUATIONS attributes in the STIFFNESS MATRIX relation.

The LOSMAT routine loaded the stiffness matrix into the STIFFNESS MATRIX DEFINITION relation. Tuples were inserted using degree of freedom values, the array containing the stiffness elements for prescribed degrees of freedom, and the upper band of the stiffness matrix stored in banded form. Although only tested with a banded "incore" solver the designed database structure for the stiffness matrix should be satisfactory for most other types of solver (for example, "Gaussian reduction on the full matrix", "out-of-core", or "skyline"), the only change necessary being the development of an equivalent to LOSMAT.

(d) The GETLOA routine retrieved load data from the database given the structure, stiffness matrix and load case set identities (refer fig. 52).

The prescribed degree of freedom numbers and identities were retrieved by invoking routine REIVD. The REIVD query was similar to the following MIMER/QL commands:

```
GET W (PRESCRIBED_DOF.STRUCTURAL_NODE,
      PRESCRIBED_DOF.DIRECTION:
      PRESCRIBED_DOF.DOF_PRESCRIBED)
WHERE PRESCRIBED_DOF.STRUCTURE_ IDENTITY EQ 'x';
GET DEGREE_OF_FREEDOM_NUMBERING.direction, W.STRUCTURAL_NODE
WHERE DEGREE_OF_FREEDOM_NUMBERING.STIFFNESS_MATRIX_IDENTITY EQ 'y'
AND DEGREE_OF_FREEDOM_NUMBERING.STRUCTURAL_NODE EQ
       W.STRUCTURAL_NODE
AND W.DIRECTION EQ 'direction';
```

For each row in the work table W, the related DOF number was retrieved from the DEGREE OF FREEDOM NUMBERING relation.

GETLOA then retrieved load data for all non-factored load cases. The factored load cases were determined by a query similar to the following:

```
GET LOAD_FACTOR.LOAD_CASE
WHERE LOAD_CASE_SET_DEFINITION.LOAD_CASE_SET_IDENTITY EQ 'x'
AND LOAD_FACTOR.LOAD_CASE EQ LOAD_CASE_SET_DEFINITION. LOAD_CASE;
```

The routine REGRAV retrieved the body loads from the relations BODY LOAD and ACCELERATION for a given load case.

```
GET ACCELERATION.VALUE, BODY_LOAD.DIRECTION
WHERE ACCELERATION.IDENTITY EQ BODY_LOAD.ACCELERATION_IDENTITY
AND BODY_LOAD.LOADCASE EQ 'x';
```

The REMEM routine retrieved member loads from the MEMBER LOAD, POINT MEMBER LOAD, FORCE, MOMENT, UNIFORM DISTRIBUTED MEMBER LOAD, UNIFORM DISTRIBUTED LOAD,and LINEAR DISTRIBUTED LOAD relations.

The first part of this routine retrieved the element number, member load type and identity for each occurrence. In MIMER/QL:

```
GET    MEMBER_LOAD.MEMBER_LOAD_TYPE,
       MEMBER_LOAD.MEMBER_LOAD_IDENTITY,
       MEMBER_LOAD.ELEMENT_NUMBER
WHERE MEMBER_LOAD.LOAD_CASE EQ 'x';
```

Then for each member load found the routine retrieved load data according to the type of member load. For example, for a point force member load a similar query to the following was used:

```
GET    POINT_MEMBER_LOAD.DISTANCE,
       POINT_MEMBER_LOAD.DIRECTION,
       FORCE.VALUE
WHERE FORCE.IDENTITY EQ POINT_MEMBER_LOAD.POINT_LOAD
AND    POINT_MEMBER_LOAD.IDENTITY EQ
MEMBER_LOAD.MEMBER_LOAD_IDENTITY;
```

Whether the point member load was a force or a moment was determined from the DIRECTION attribute.

The REPRDI routine retrieved prescribed displacements from the PRESCRIBED DISPLACEMENT relation. Its query was similar to:

```
GET    DISPLACEMENT_VALUE.VALUE
WHERE PRESCRIBED_DISPLACEMENT.LOADCASE EQ 'x'
AND    PRESCRIBED_DISPLACEMENT.DOF_PRESCRIBED EQ 'y'
AND    PRESCRIBED_DISPLACEMENT.DISPLACEMENT_IDENTITY EQ
                    DISPLACEMENT_VALUE.IDENTITY;
```

The values for y were obtained in the earlier call to the REIVD routine.

The REFOR routine retrieved nodal forces from the NODAL LOAD, MOMENT and FORCE relations. For forces the query was similar to:

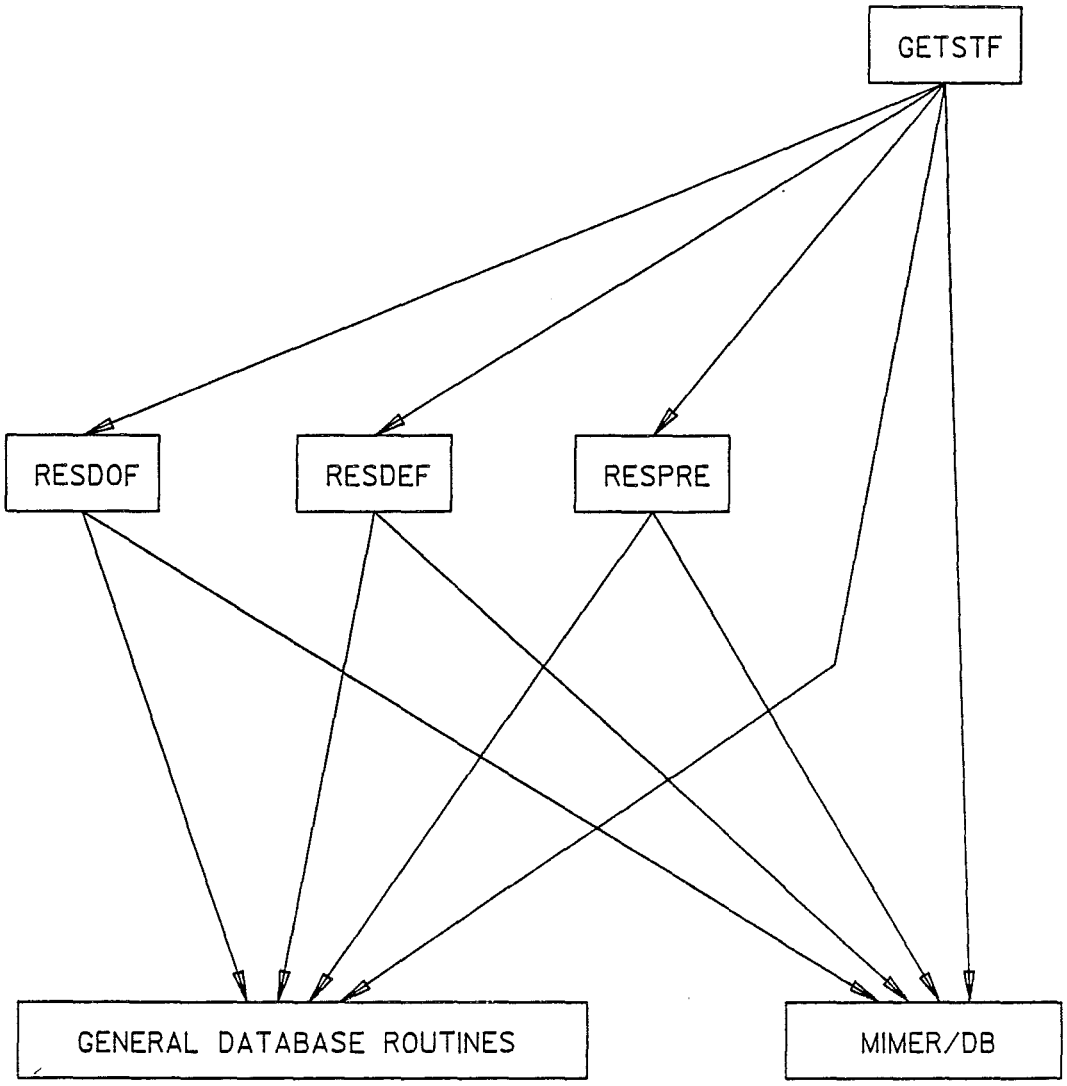


Figure 53. Structure Chart for GETSTF Routines

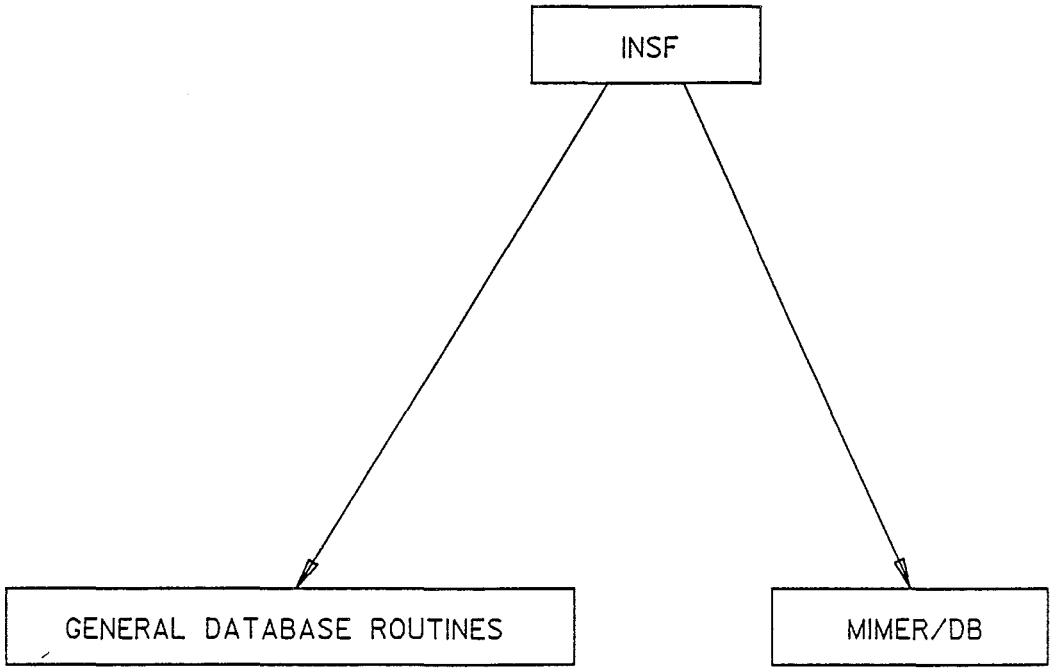


Figure 54. Structure Chart for INSF Routines

```
GET   NODAL_LOAD.STRUCTURAL_NODE, NODAL_LOAD.DIRECTION,
      FORCE.VALUE
WHERE NODAL_LOAD.LOADCASE EQ 'x'
AND   NODAL_LOAD.POINT_LOAD EQ FORCE.IDENTITY;
```

Again whether the load was a force or moment was determined from the DIRECTION attribute.

(e) The GETSTF routine retrieved the stiffness data for a given stiffness matrix. Its structure chart is figure 53.

The RESDOF routine retrieved the degree of freedom numbers from the DEGREE OF FREEDOM NUMBERING relation. Its query was similar to:

```
GET   DEGREE_OF_FREEDOM_NUMBERING.DX,
      DEGREE_OF_FREEDOM_NUMBERING.DY,
      DEGREE_OF_FREEDOM_NUMBERING.DZ,
      DEGREE_OF_FREEDOM_NUMBERING.RX,
      DEGREE_OF_FREEDOM_NUMBERING.RY,
      DEGREE_OF_FREEDOM_NUMBERING.RZ,
      DEGREE_OF_FREEDOM_NUMBERING.STRUCTURAL_NODE
 WHERE DEGREE_OF_FREEDOM_NUMBERING.STIFFNESS_MATRIX EQ 'x';
```

The RESDEF routine retrieved the SEMI-BANDWIDTH, NUMBER OF DOF, and TOTAL EQUATIONS attributes of the STIFFNESS MATRIX.

The RESPRE routine retrieved the stiffness element values for prescribed degrees of freedom from the STIFFNESS MATRIX DEFINITION relation.

(f) The INSF routine (refer fig. 54 for its structure chart) loaded the right-hand side matrix into the ASSEMBLED R.H.S. DEFINITION relation. It inserted a tuple for each value in the matrix. In addition to the matrix and DOF numbers it was passed the right-hand side

matrix identity and a vector of load case identities generated in routine GETLOA.

6. THE GLOBAL CONCEPTUAL VIEW

The local views were merged one at a time into a global conceptual view of the database. This process starts with merging the data element definitions, testing for synonyms, homonyms, and so forth. Then the data element dependencies must be merged, identifying any new dependencies and rationalizing duplicate dependencies. The data aggregates can then be formed and tested using the same tests applied to the local views.

No changes were found necessary in forming the global view, other than adopting consistent units (that is, metres, kilograms, seconds, and Newtons) for the data elements. Therefore the global view is the union of the data element definitions, dependencies, and aggregates for the local views in appendices VI, VII and VIII.

7. MAPPING THE CONCEPTUAL VIEW TO THE LOGICAL VIEW

The global schema was mapped into a logical schema in terms of the DDL of the selected commercial DBMS, MIMER.

(1) Selection of DBMS

The relational MIMER system was used because it was the only DBMS available at the University of Canterbury at the time of undertaking this study. The relational approach does however have many advantages over the network model in this type of application.

It has a strong mathematical foundation and is simple and flexible in nature. Relational entities can be added or deleted, attributes added or deleted with no effect on unrelated data. The wagon designer can view the data in any desired form without worrying about pointers and predefined access paths. These views can be defined as work relations and accessed as though they were relations. Views, constraints, and indices can be defined or dropped at will. As a result, changes to the schema may be made without harming application programs that use the data (other than in performance). Alternatively changing the schema of a network database can mean recompiling the database definition, halting the entire database system, and reloading the data where physical pointers have been affected.

There are few, if any, instances of grouping together distinct concepts in the relational approach, unlike the network approach where the same data construct can be used to carry association information, an access path, and certain integrity constrains (Date, 1981, chapt. 28). In a relational schema any of the above may be changed independently from the others with minimal effect on programs and users, whereas in the network case a change in say an integrity constraint could affect programs that have been relying on the construct for say an access path.

Relational systems are simple and easy to use as there can be a number of ways of linking tables to access data - navigation links are transparently obvious and are defined at the time of accessing the data.

In engineering design, the data and the schema are usually in a state of flux. Unexpected new objects and relationships are integrated as needs dictate, and initially there is little understanding of the objectives. Relational systems have the modification flexibility which

allows changes to take place without seriously disrupting the existing applications. The network model, in contrast, suits well understood environments with plain, logical boundaries. In this environment network DBMS's perform with speed, efficiency and consistency.

Relational DBMS's have the major drawback that they require a lot of system resource for searching and sorting. The performance of relational DBMS's depends on the physical schema not the logical schema, and if the physical schema is tailored to usage patterns then, it is claimed, performance should be comparable to the other data models. Efficient access methods and optimised code can alleviate this problem and now hardware processors are being offered that are specially designed to speed up database management operations. They range from peripheral devices for speeding up I/O to processors that completely handle database management functions.

(2) Introduction to MIMER

MIMER could be run in multi-user or single user mode. Under multi-user mode the different MIMER/QL users as well as other application programs shared common resources such as MIMER processes, and system databanks. The multi-user system also provided synchronization of operations.

In order to preserve data integrity during concurrent access to a shared database, update operations were grouped into indivisible units of operation on the database (called transactions) that transformed a consistent data state into a new consistent state. MIMER used methods called "optimistic concurrency control" and "careful commitment" to implement concurrent accessing. (There was a restriction that all

operations making up a transaction had to take place within the same databank.) The logging facility kept a record of committed transactions on a separate logging media. This logging media could then be used together with a backup copy of the database to restore the database status up to a desired point if a hardware failure occurred. The log file could also be used to obtain information on how the database had changed with time.

In a single user system concurrency control was not in operation, but transaction management could still be used.

In MIMER, access control and allocation of storage was defined with storage units called databanks. A databank could contain several tables and any given table was wholly contained within a single databank. Each databank also contained an active data dictionary (a description of the tables contained in the databank) in the form of a table. A database could contain an arbitrary number of databanks. The DBA command was provided in MIMER/QL to define databanks, and grant user access privileges as well as to define users.

The data definition and manipulation facilities as well as the database administration facilities were available in MIMER/DB loosely coupled to programming host languages through routine calls. Dynamic construction of requests and cursor operations were possible.

(3) The Implementation using MIMER

MIMER was implemented using Burroughs "library" facilities. A library is a program which provides a set of procedural "entry points" which can be called on by other user programs. When a library entry point is first called, the executing user program is suspended and the

library program runs until the exported entry points are made available at which point they are linked with those declared in the user program and the user program resumes execution. A library may itself function as a user program, and call other libraries and export procedures that are declared as an entry point of another library, thereby providing indirect linkage. Simultaneous use of a library is either private or shared. With shared libraries, the changes made by any user program to global items in the library apply to all user programs that call that library. The private option creates a separate library stack for each user program that calls the library.

Only the single user mode was available on the Burroughs at the time this work was undertaken.

The tables implemented in the MIMER system are presented in appendix X along with descriptions of the tables and columns.

Some changes were made to the conceptual schema in order to form the MIMER system logical schema. Most were of a relatively minor nature such as changing relation names and attribute names to fit the eight character maximum for table and column names. There was no variable length text field specification available so the manufacturing specification text had an 80 character format and an attribute called line number was added to the primary key.

Better performance in searching for a correspondence file number for a particular item may have been achieved if tables relating identifier with file number were defined for each type of item (for example, assembly file and part file tables). However this approach would increase table overheads. The difference in performance between the implemented item file approach and part file, assembly file approach

would depend on the specific access methods used (for example, sequential or secondary index).

The addition of the ITEM TYPE relation permits determination of whether the item identified in the correspondence file, section number, manufacturing specification, and assembly sub-items relationships is to be found in the ASSEMBLY or PIECE PART relation. This determination could also have been made by searching both the ASSEMBLY and PIECE PART relations. By including the type definition table for items and geometrical assemblies all the relevant tables do not have to be searched to find the item attributes given a correspondence file number; one can go directly to the table specified. This approach would seem to be quite general. If the uniqueness constraint on item identifier or geometrical assembly number were relaxed then there would have to be separate tables for each type.

8. REFERENCES

BUCHMANN, A.P. (1981) Databases for Process Plant Design. In Leesley, M. Computer-aided Process Plant Design. Gulf Publishing. p.149-189.

DATE, C.J. (1981) An Introduction to Database Systems. 3d.ed. Reading, Massachusetts, Addison-Wesley. 574 p.

CHAPTER XVIII

DESIGN AND IMPLEMENTATION OF THE PROTOTYPE RUN-TIME INTERPRETER

This chapter describes the run-time interpreter that was designed and implemented specifically to execute the code generated from commands in the language presented in chapter XVI. (Run-time refers to the time at which the code is interpreted.) The interpreter was stack oriented and based on the one described by Tremblay and Sorenson (1982). Many of the object language instructions were similar to Tremblay's and so the reader is referred to appendix XI for details on the instructions. A more detailed description of the instructions relating to the solve command is given in section 2. As stated previously only enough of the interpreter was implemented to illustrate the benefits of the proposed computer aided design scheme.

1. RUN-TIME ENVIRONMENT

The code area was implemented as a one dimensional array with the location in this array of the current instruction held by a variable called IC. The initial instruction was stored in the second location of the code array as the first was reserved for error handling and contained the stop instruction.

The data area was made up to two arrays: an integer array containing the character strings and lengths stored contiguously and the

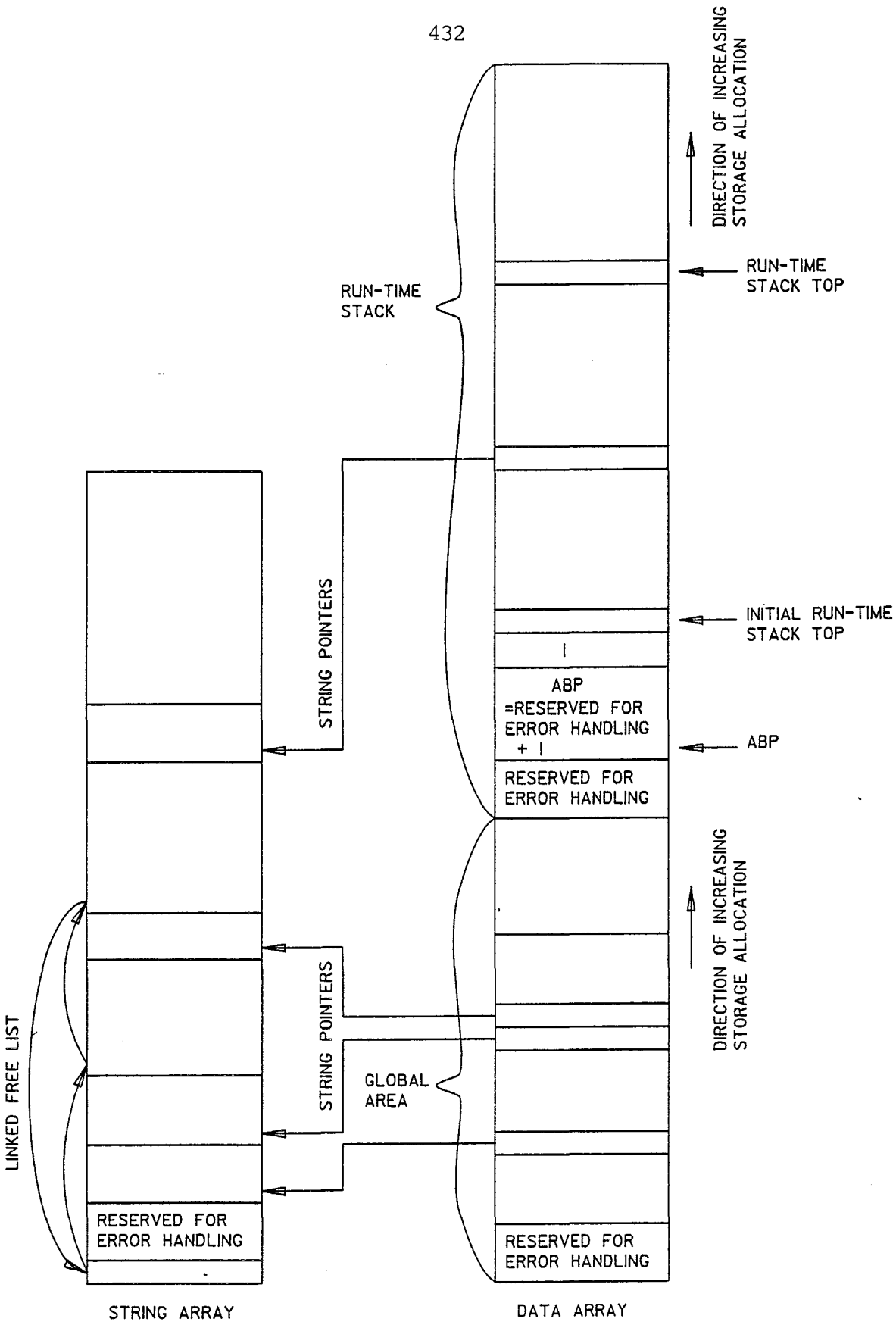


Figure 55. Structure of Data in the Run-time Interpreter

linked list of free blocks; and a real array for the global variable storage and the run-time stack. Variables were used to determine storage spaces allocated for the strings and variables (refer fig.55).

The run-time stack was used for temporary storage of values while commands were executed. Each intermediate code instruction accessed the top most locations, deleted zero or more of them and finally pushed zero or more values onto the stack. If a negative string pointer was encountered, then the string was temporary and was removed from the string area when the operation had finished with it.

An activation record containing contiguous space was created on the run-time stack whenever a solve command was executed. This space was freed when the solve command finished its processing or when it was terminated by the user or an error condition. The activation record was built up initially by the execution of the code generated directly from the solve command. This phase built the segment illustrated in figure 46 which contained the stated parameters, a return address and the current value of the activation base pointer. The next stage began with the resetting of the activation base pointer. Data was then added to the record as it was retrieved from the database. The procedure for performing the calculation was then called and finally data was added to the database before the solve command finished executing.

2. IMPLEMENTATION OF THE RUN-TIME INTERPRETER

(1) Software Organisation

The hierarchy of the procedures comprising the interpreter is shown in figure 56. EXECUT initialised the end and suspend execution flags

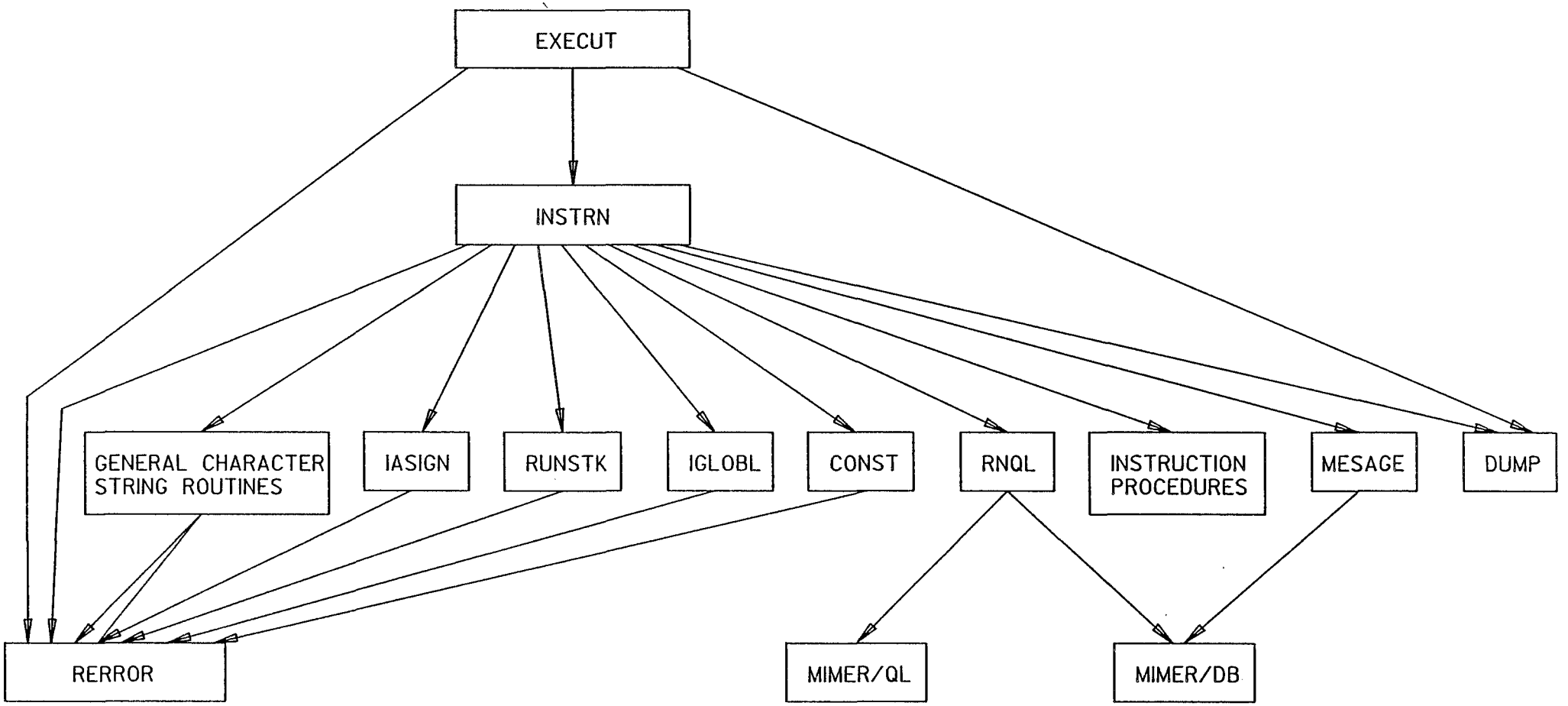


Figure 56. Structure Chart of the Run-time Interpreter

then called INSTRN repeatedly to execute the instructions until the suspend or end flags were set indicating a suspended solve command, completion of a command or an unrecoverable error condition. EXECUT could also print out the code and data areas for debugging purposes.

The procedure INSTRN interpreted the instruction in the code array, whose address was given by the value of the current instruction counter variable. It checked if the next instruction could be executed, if it could, it did so and incremented the instruction counter (unless an error occurred during execution, then the instruction counter was set to the top of the used code area) and finally the routine returned. A computed GOTO statement which used the coded value of the current instruction (emulating a CASE construct) was used to select the code to perform the instructions' operations.

The procedures IASIGN, RUNSTK, IGLOBL checked the bounds of the global area and run-time stack before access to that area was attempted, returning either a valid index or the value at a validated location. The procedure CONST returned the value at a validated location in the temporary real array. This temporary array was used to pass real numbers to the run-time interpreter from the command interpreter. RERROR printed error messages and set the end flag if required, and DUMP printed the run-time stack and some variable values. MESAGE output informational messages or requests to the designer. It called some machine dependent routines in the MIMER/DB library to perform output without linefeed. RNQL initiated a MIMER/QL session.

(2) Instruction Procedures

This section describes the procedures used to implement the

instructions generated from the solve command. Details of these routines can be found in appendix XI.

(a) Get Default Method. The instruction before the instruction to get the default method pushed zero on to the top of the run-time stack. This value signalled to the GDFALT routine that the instruction was not suspended. The routine then attempted to retrieve a method from the DFALTMET table in the DATADICT databank for the model and all the dependent variables listed in the activation record. It did this by searching through the default methods for a method stored for the specified model and the first dependent variable that was also stored for all the other dependent variables (and the specified model). The first default method found that produced the dependent variables listed in the activation record was stored in the activation record and its negated string pointer replaced the zero on top of the run-time stack. A message prompted the wagon designer to accept the default or enter an alternative method and the suspend flag was set.

If no default method was found, the error handling routine was called and the procedure exited.

If the value on top of the run-time stack indicated the instruction was suspended and the wagon designer had provided an answer to the method prompt, then the suspend flag was reset and the answer stored in the method specification slot of the activation record.

If the value on top of the stack indicated a suspended instruction but no answer provided, then the suspend flag was set and the wagon designer was prompted again using the default value previously determined and stored in the activation record.

On return to the calling routine, if the suspend and error flags

were not set, the instruction counter was incremented and the communication location popped off the top of the stack. If an error was detected, the string storage locations referenced in the activation record were de-allocated, the error and suspend flags were reset, and the return and stop instructions were pushed on to the top of the code stack with the instruction counter set to execute these instructions next.

The structure chart of the default method routine is presented in figure 57. MIMERR is the routine that handles error return codes from the MIMER/DB routines, otherwise the routines are as described above.

(b) Check Validity of Method, Model and Variables. The CHKMET routine checked that the variables were defined in the database, and that the specified method was a valid way of calculating the dependent variables in the specified model. It also checked for any additional dependent variables calculated by the method but not specified by the wagon designer.

The model, method and dependent variable names were taken from the activation record. The NAMEDICT table in the DATADICT databank was used to retrieve the variable numbers for both the supplied dependent and independent variables. If no entry was found the run-time error handler was invoked. Following successful retrieval, the variable names were exchanged for the variable numbers and the respective string storage locations freed.

The SOLUTDEF table was used to obtain the solution number for the combination of model, method, and the dependent variables. If no rows were found, it was reported to the wagon designer that the system could not solve for this variable using the specified method on the specified

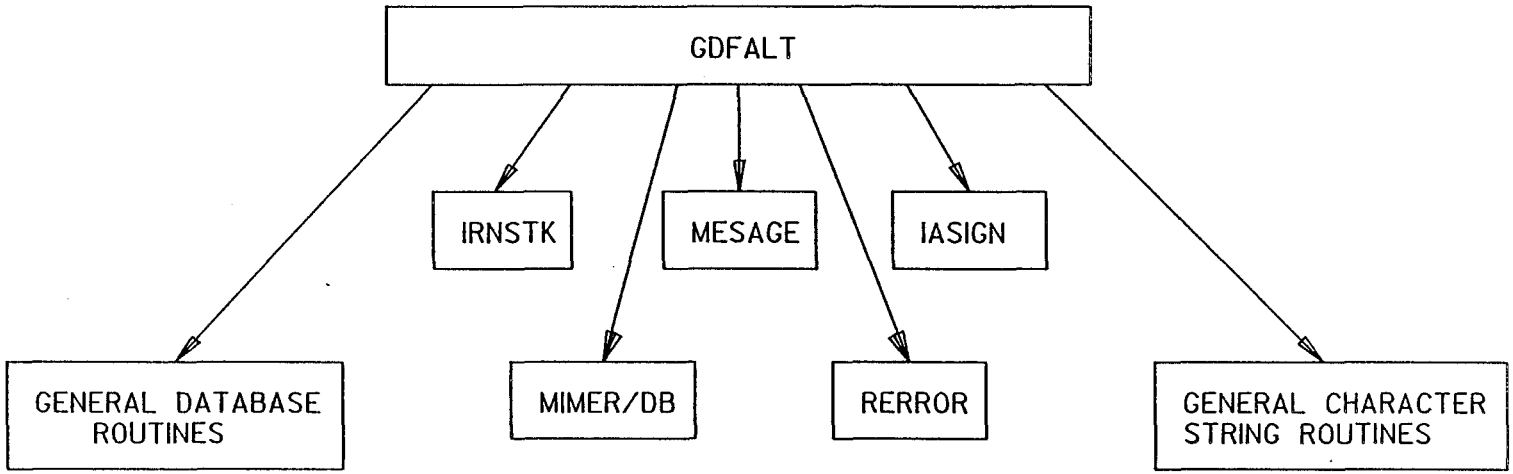


Figure 57. Structure Chart for the GDFALT Routines

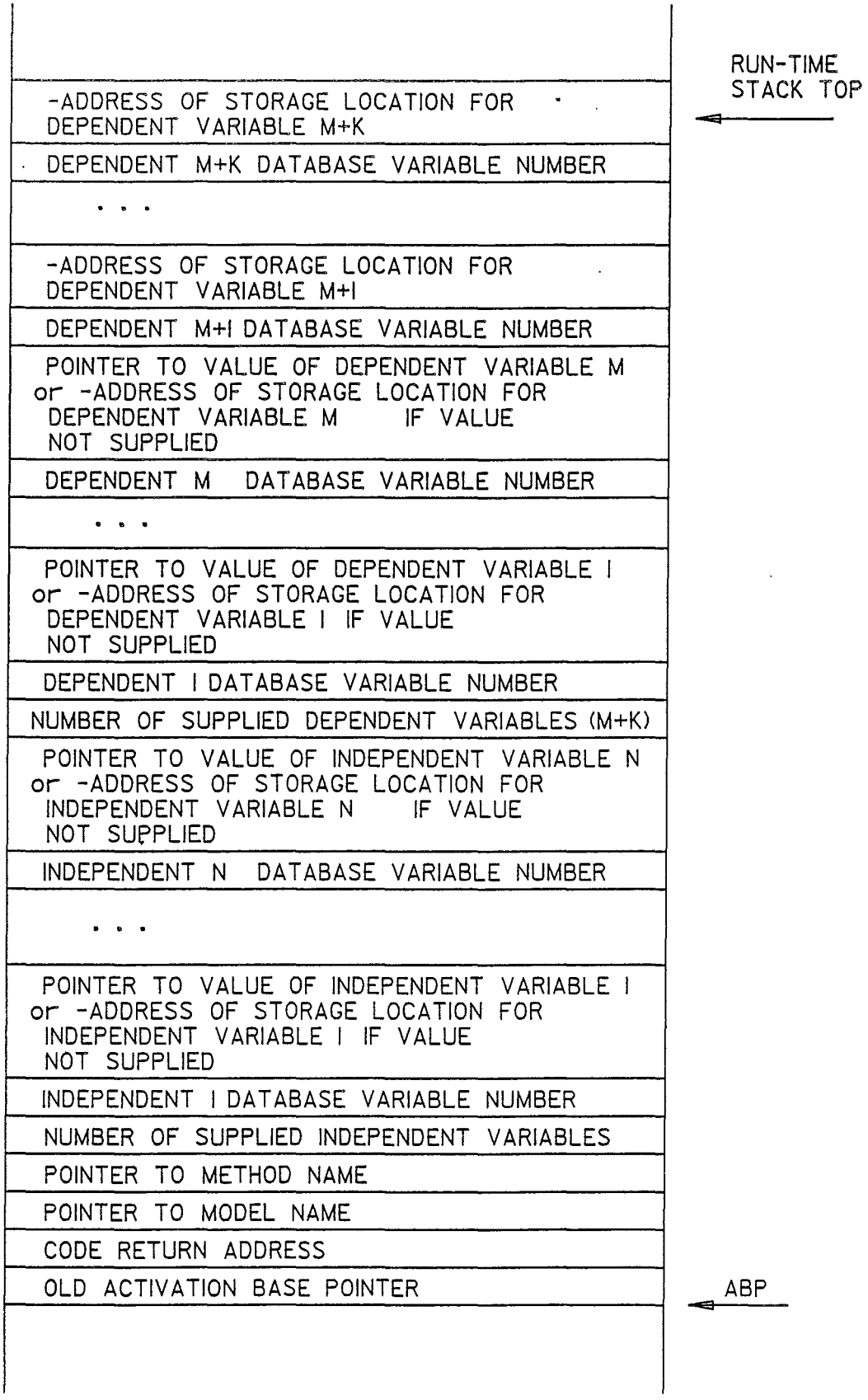


Figure 58. The Activation Record on Exit from the CHKMET Routine

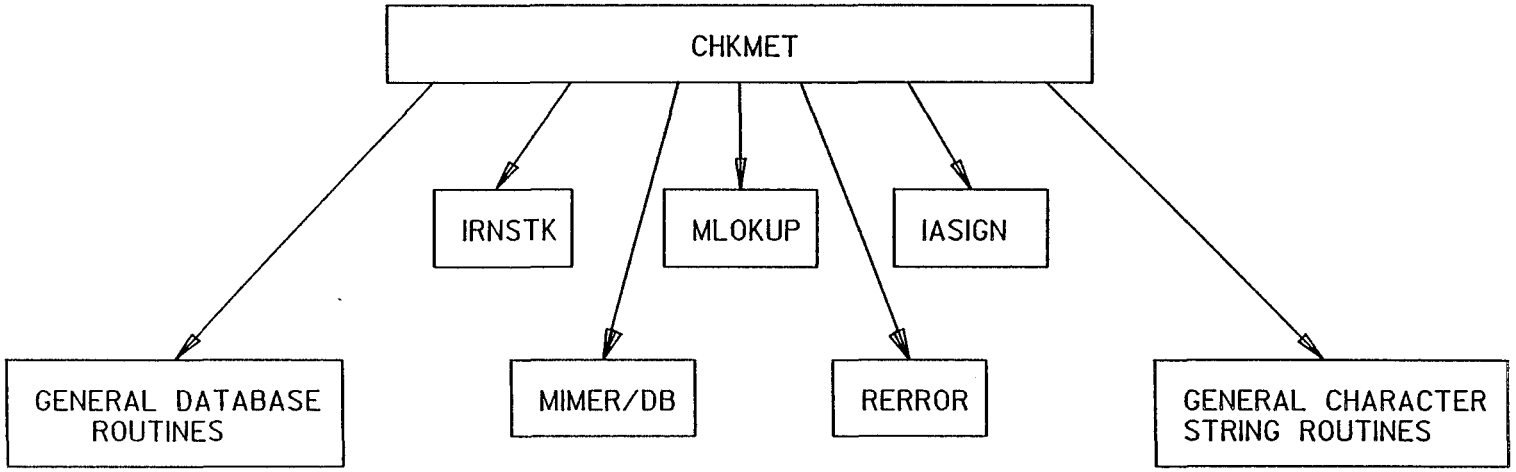


Figure 59. Structure Chart for the CHKMET Routines

model, and the error flag was set.

Using the solution number found in the retrieval(s) described above, and the model and method names, the SOLUTDEF table was searched for dependent variables other than those already supplied. If other dependent variables were found, their variable number was obtained from the NAMEDICT table and the variable number together with its storage address in the run-time stack were pushed on top of the activation record. The storage address was found by a lookup in the symbol table. The storage address was negated in the activation record to indicate that it was an address not a string pointer. When there were no more dependent variables, the total number of dependent variables was updated in the activation record.

Before the routine returned to INSTRN, the instructions to get the model data, independent variable instance numbers, dependent variable instance numbers, and to check the presence of the dependent variables, were inserted into the code area.

On return to INSTRN the instruction counter was incremented unless an error had occurred, in which case it was handled in the same manner as for the GDFALT routine.

On successful exit from CHKMET the activation record was as presented in figure 58.

Figure 59 illustrates the hierarchy of routines invoked by CHKMET.

(c) Retrieve Model Data. The GETMD routine retrieved information on the model from the database.

Firstly it obtained from the MODELDIC table in the DATADICT databank the model table name and databank name for the model named in the solve command. The pointers to the two name strings were pushed on

top of the action activation record.

Then the column names in the model table for the respective dependent variables were retrieved from the COLUMNS table in DATADICT. Using this information the column numbers were obtained for these columns from the MIMER table definition table *TABDEF in the databank where the model table resided. The column names were stored in a string buffer, ordered according to column number. The column numbers were placed on the activation record. If no row was found in either of these table retrievals, then GETMD enforced a database design constraint by setting the error flag and calling the database constraint error handling routine MIMCON.

The column names and column numbers in the model table for the independent variables were then retrieved from the *TABDEF table. All columns were selected except those with a column number equal to any dependent variable column number or with column name equal to "METHOD". Having previously retrieved from the SOLUTDEF table all dependent variables calculated using the specified method on the model, the remaining columns in the model table were independent variables. This was a generic property of model tables. The column names and numbers were stored in the same manner as the dependent variables.

Following retrieval of the independent variables' column information, the number of independent and dependent columns and the pointer to the column names' string were stored in previously allocated space in the activation record.

Before returning to INSTRN, space was allocated for database information on the dependent variables, variable instance identity values, and a column counter. The number of independent variables was

stored in the column counter location and a one was placed on to the top of the activation record.

The actions on return to INSTRN were as described for the CHKMET routine.

Figure 60 presents the structure chart for the GETMD routine.

(d) Get the Independent Identities. The GETIID routine retrieved from the database the identity numbers for the independent variables.

The activation record before GETIID was called is illustrated in figure 61.

If the column counter value was zero on entry to GETIID then there were no more independent variables left. The instruction counter was increased, the stack top decreased by one, and the column counter assigned the number of dependent variables, before the routine exited.

Otherwise, the variable number was retrieved from the COLUMNS table for the current variable (as determined from the column counter), for the model table, model databank, and model table column names (as stored in the activation record). If a row did not exist with these attributes then the database constraint handling routine MIMCON was called. The complete list of independent variables as determined from the GETMD routine was used in this routine as opposed to the list of supplied independent variables because not all independent variables need be supplied. (Moreover the prototype language did not require the specification of any independent variables in the initial command.)

If the communication area on top of the activation record had a value of zero then the instruction had not been suspended. This value was popped off the stack after being read. The routine then tested to see if the independent variable had been supplied, by comparing the list

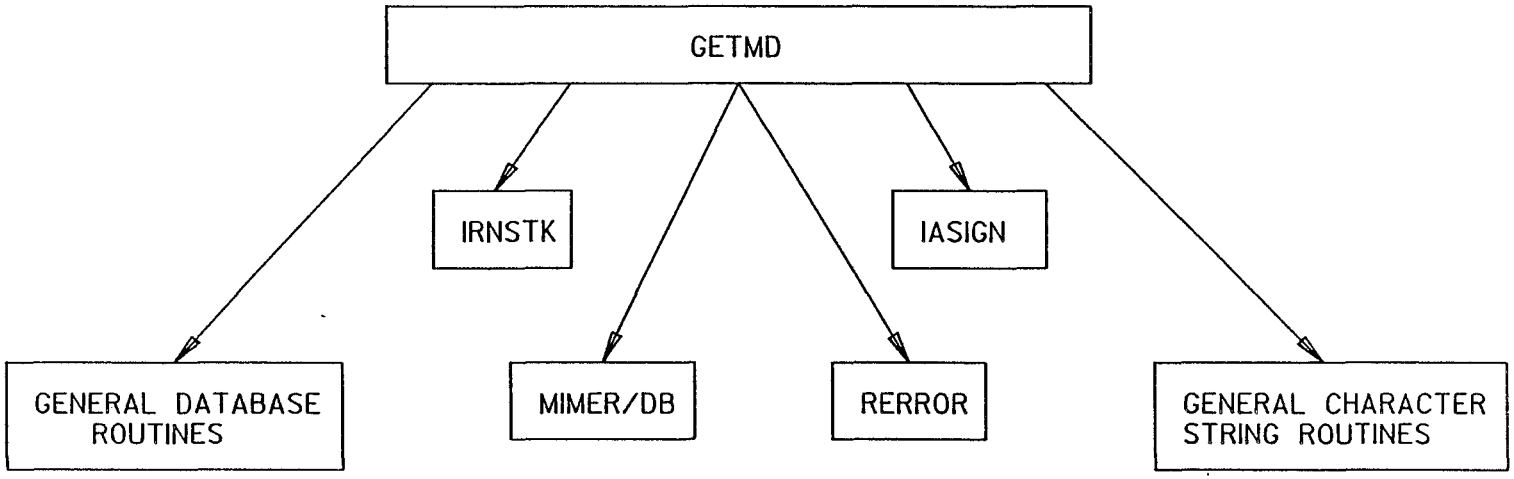


Figure 60. Structure Chart for the GETMD Routines

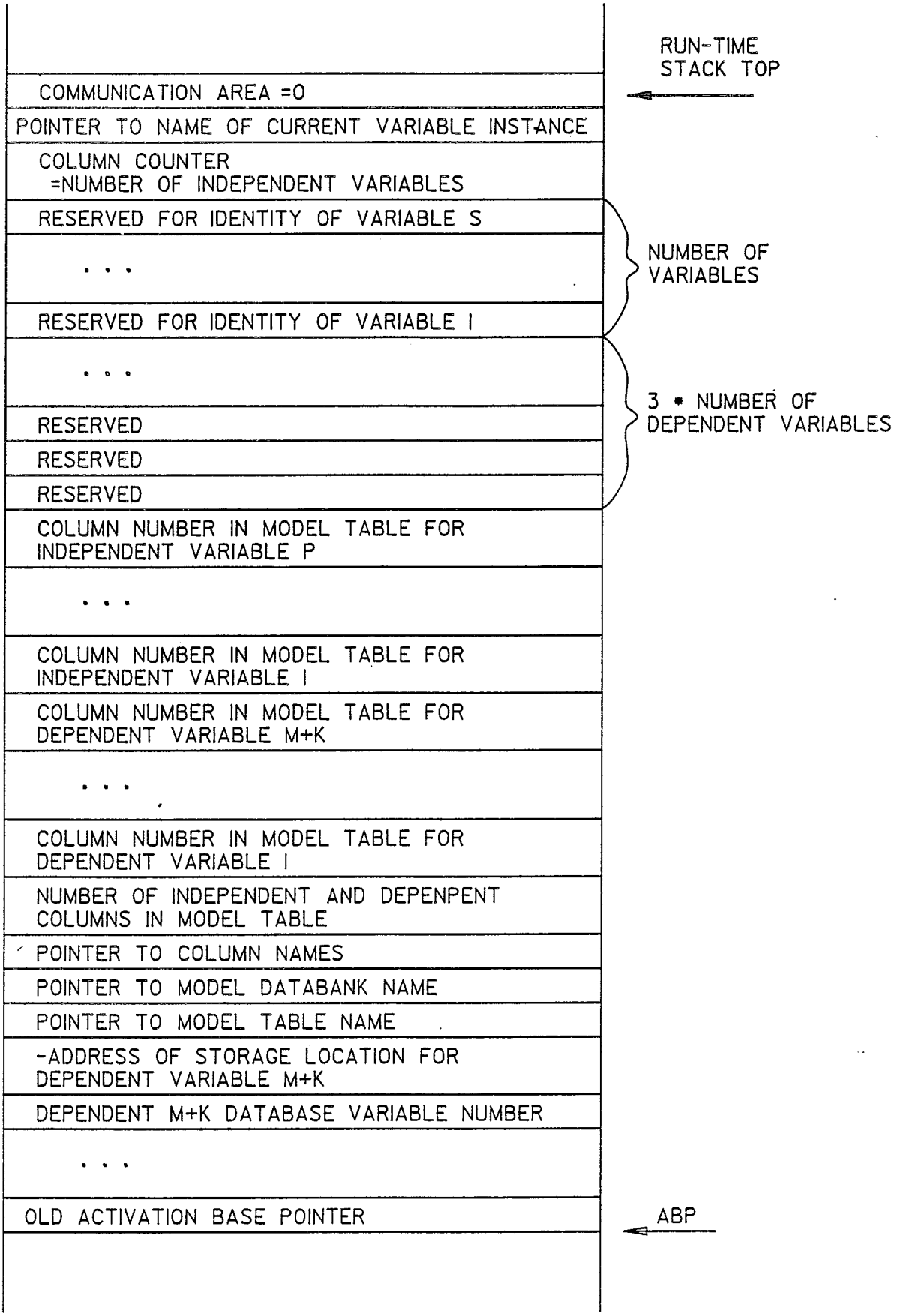


Figure 61. Activation Record on Entry to GETIID Routine

of supplied independent variable numbers with that obtained from the earlier database retrieval of variable number for the variable under consideration. If the independent variable was in the command line, the activation record was checked as to see whether the instance name had also been supplied (which was true if the instance name location held a positive value).

If the instance name had not been supplied then the variable name was retrieved from the NAMEDICT table using the current variable number. Again MIMCON was called if no row existed. The MLOKUP routine was called if the independent variable was not in the command, to supply the storage location in the stack of the variable. The suspend flag was set and the pointer to the current string value (that is, the default value) of the variable was placed in the current variable instance name location of the activation record. This string pointer was also pushed onto the communication area with a negative sign indicating the instruction had been suspended. Finally the wagon designer was prompted to confirm the default or supply another value before the routine exited.

If the instruction had been suspended and no answer had been given to the prompt for the instance name and the answer required was a name and not a number (as indicated by a positive value of the name pointer of the current variable instance) then the actions above were also executed.

If the instance name had been given, either supplied in the command or given as an answer to the suspend prompt (indicated by a value greater than zero in the communication area) then the instance number was retrieved from the database. The instance name pointer was placed

in the current variable instance name pointer location in the activation record, and the default string storage freed if unused. The name table, name table databank, and variable name were obtained from the NAMEDICT table using the current variable number. MIMCON handled the no row condition.

The GETCOL routine then retrieved the column name of the first column (the instance number column by definition of the generic instance name table) of the name table from the *TABDEF table in the name table databank.

The instance number(s) were retrieved from the name table for the instance name and non-null model values. If an answer had been provided to the message requesting selection of an instance from a list of instances with the specified name (indicated by a positive value in the communication area and a negative value in the current variable instance name pointer location) then the number of instances found was set to one. The independent variable was given the i'th instance number in the list as determined from selection of item i in the displayed list.

If only one independent variable instance was found with the specified name, its instance number was stored in the activation record, the column counter decreased by one, and zero pushed on top of the stack. The routine then exited.

If no independent variable instances were found, the suspend flag was set, and the negated current instance name pointer pushed on top of the stack (that is, the communication area). A message indicating the status was displayed to the designer, then the routine exited

If more than one instance was found with the same name, the suspend flag was set, and the negated instance number of the last instance found

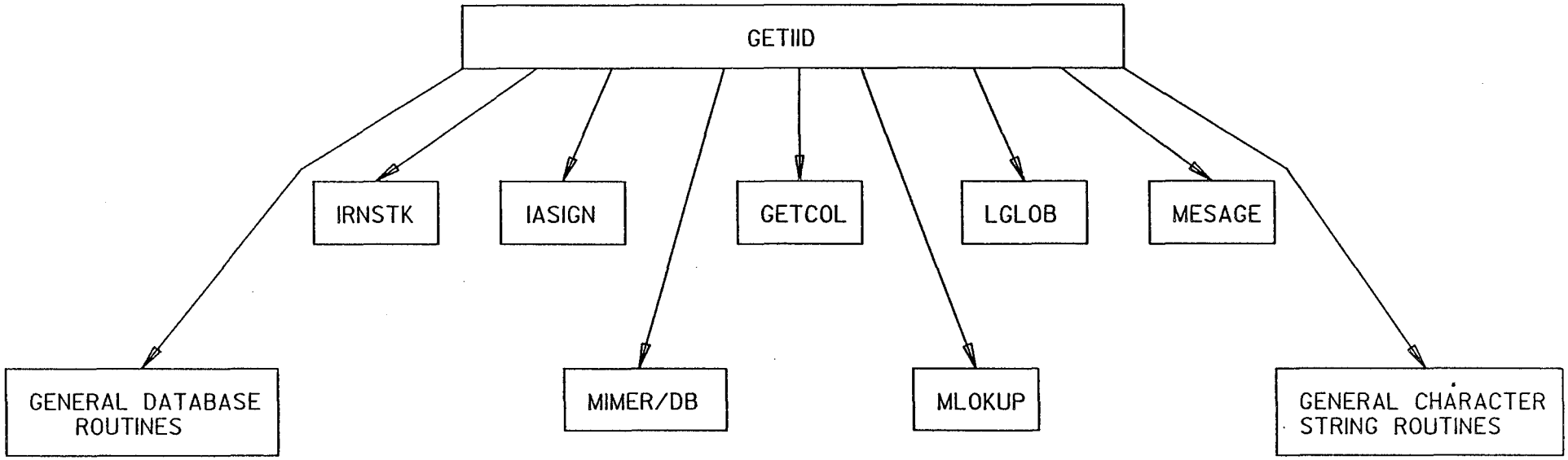


Figure 62. Structure Chart for the GETIID Routines

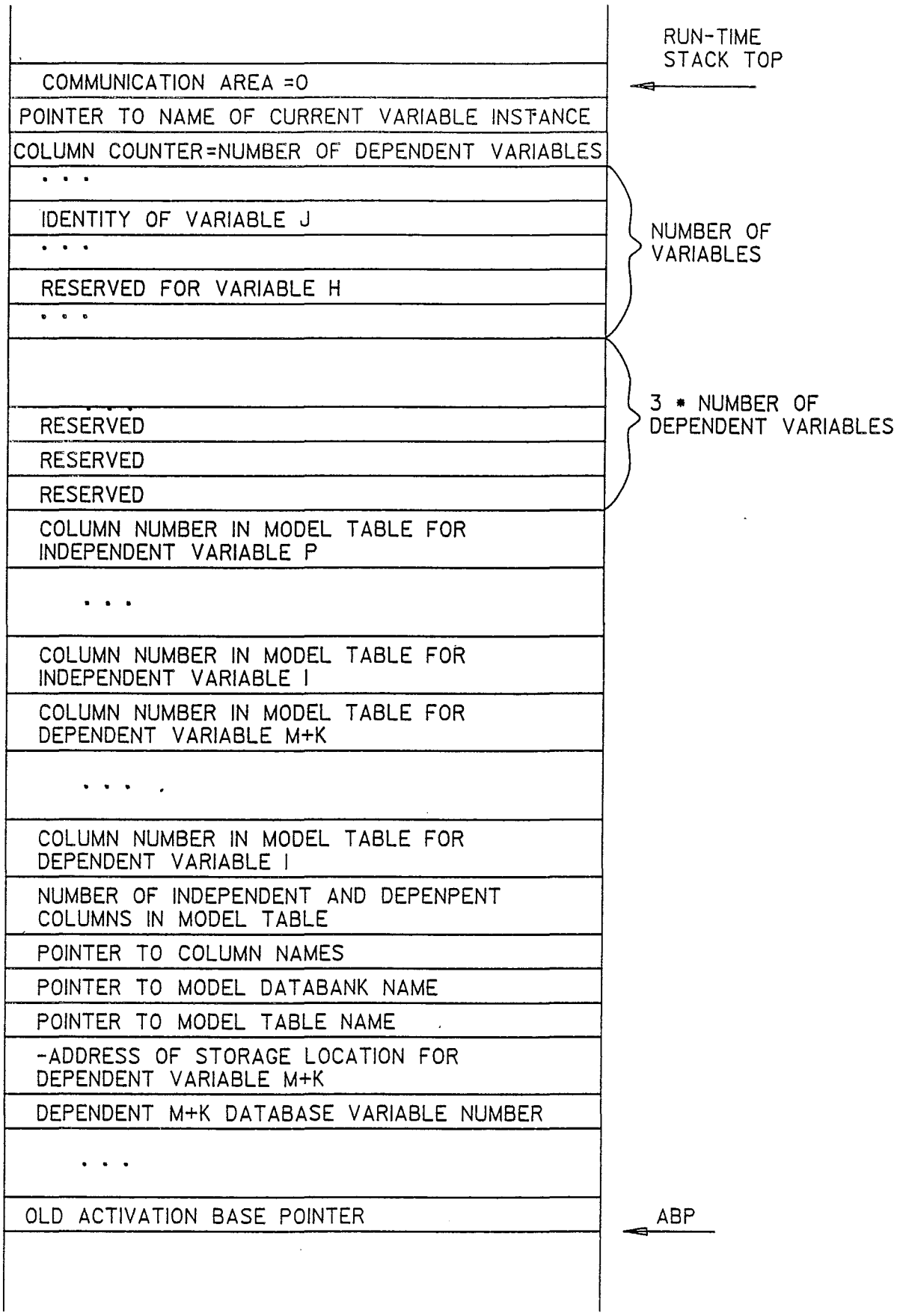


Figure 63. Activation Record on Entry to GETDID Routine

pushed onto the communication area. The negated instance name pointer was placed in the current variable instance name pointer location indicating that the communication area contained a number and that there was more than one instance with the specified name in the database. For the list of instances, the instance name, source model, and creation date were retrieved from the name table and displayed to the wagon designer. The routine prompted the wagon designer to select one instance before exiting.

If the routine had been suspended and the name pointer of the current variable instance was negative then the section dealing with retrieval of instance numbers was executed. This occurred whether or not an answer had been supplied to the prompt.

On return to INSTRN the error condition was handled as for the GDFALT routine otherwise INSTRN was exited normally.

The structure chart for GETIID is shown in figure 62.

(e) Get the Dependent Identities. The GETDID routine retrieved from the database the instance identifying numbers for the dependent variables.

Before GETDID was called the activation record was as presented in figure 63.

If the column counter was zero then there were no more dependent variables left. The instruction counter was incremented and the communication area popped off the stack before the routine exited.

Otherwise the column number and variable number were obtained from the activation record for the column under consideration.

If the value of the communication variable was zero then the routine had not been suspended. The communication area was popped off

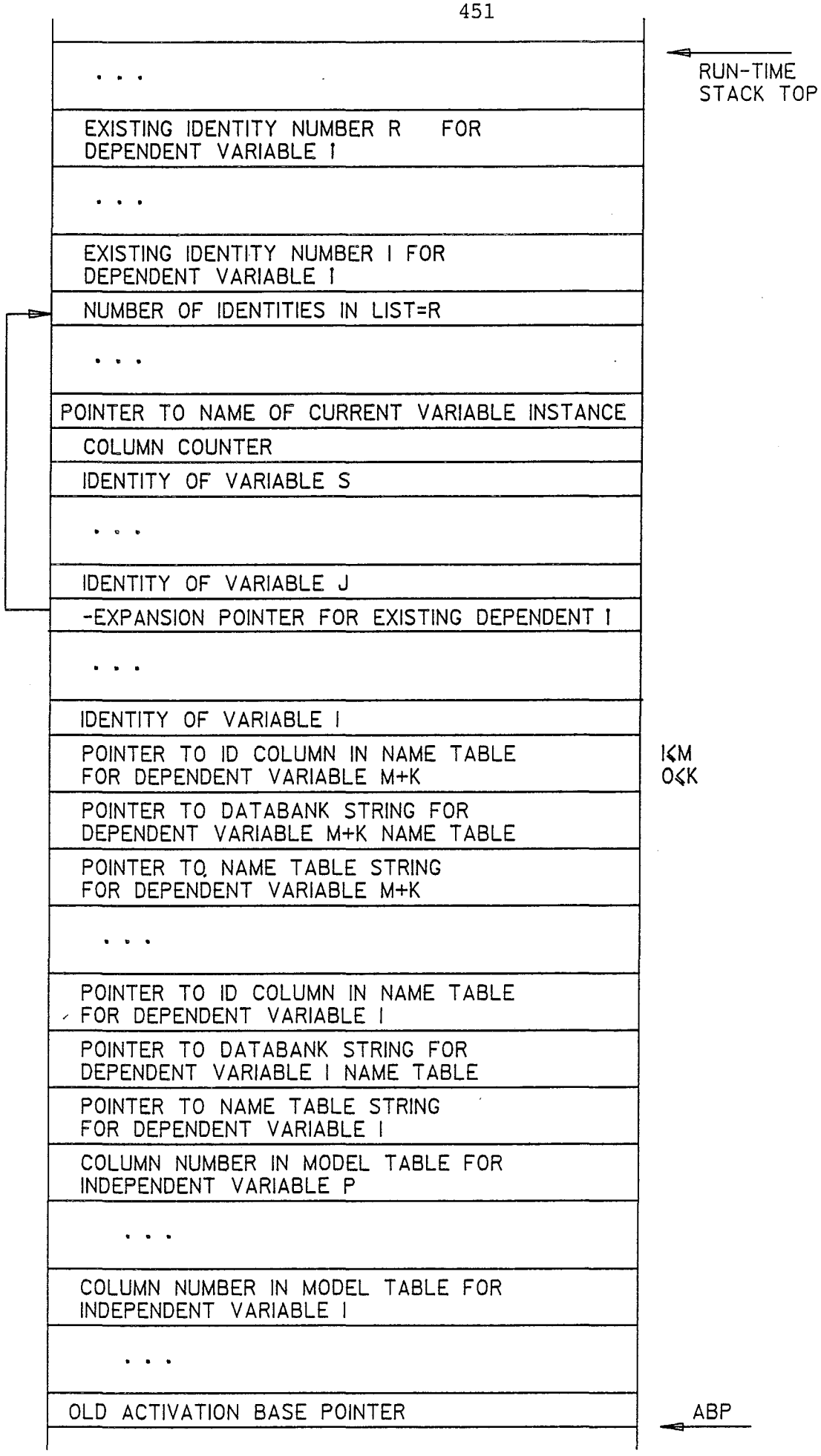


Figure 64. Activation Record on Entry to CHKPRE Routine

the stack once read. The sign of the instance name location was tested to determine whether the instance name has been supplied in the command. If the instance name had not been supplied, it was handled in much the same manner as in the GETIID routine; except no symbol table lookup was necessary. These actions were also performed if the routine had been suspended and no answer returned.

If the instance name had been given in reply to the prompt (indicated by a positive value in the communication area) then the default string storage area was freed if the default was not used and the name was stored in the activation record above the variable number.

When the instance name had been supplied by the wagon designer, the instance number was retrieved from the database. The instance name table and databank were retrieved from the NAMEDICT table for the current variable number. If no rows were found, MIMCON handled the constraint error. The GETCOL routine retrieved the instance number column in the instance name table. The string pointers to the name table, name table databank, and instance number column were stored in the activation record (refer fig. 64). Then the instance identifying number(s) with attributes of the specified name and model (obtained from the activation record) were retrieved from the instance name table. If no instance was found, the INSID routine was called to create an instance. This routine inserted a row with an instance number one greater than the existing number of rows in the table. The primary key, the instance number, was thereby unique for every instance created by the program (since deletions were not implemented). After successful creation of an instance, the identity number was stored in the activation record. The column counter was decreased by one, the suspend

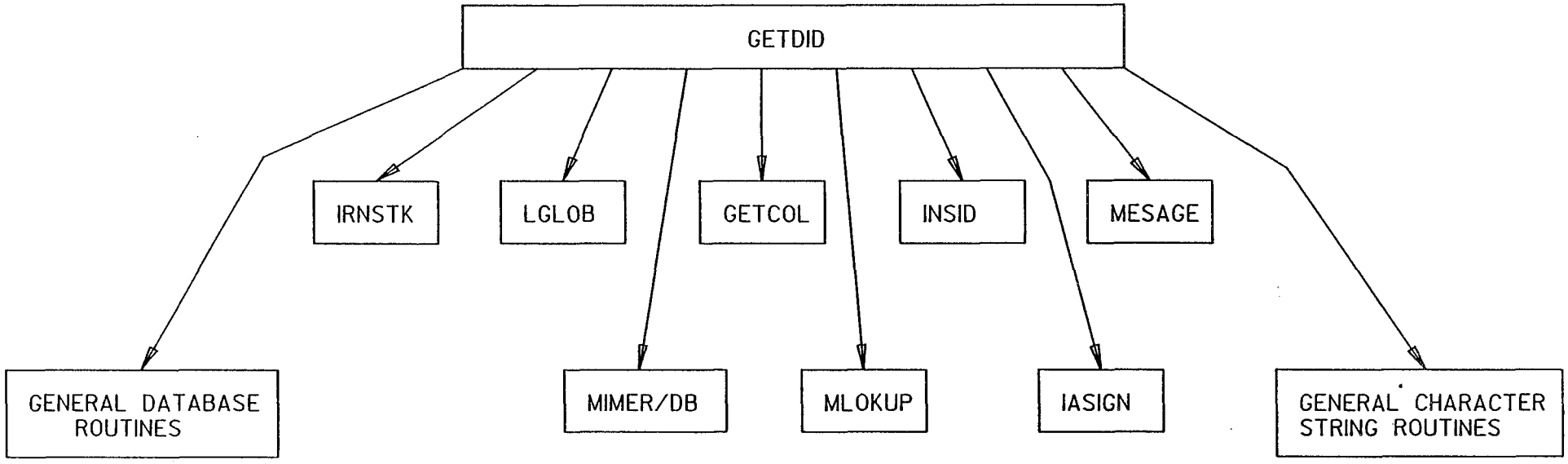


Figure 65. Structure Chart for the GETDID Routines

flag reset, and zero was pushed on to the top of the stack before the routine exited.

If an instance(s) was found then the number found was pushed on to the top of the stack and the stack pointer to this location was placed in the instance number location for the current variable in the activation record. Then the instance number(s) was pushed on to the top of the stack. Then column counter was decreased by one, the suspend flag reset and zero pushed on top of the stack (as the new communication area). The routine then exited.

The return to the INSTRN routine was handled in the same manner as for the GETIID routine.

The structure chart is presented as figure 65.

(f) Check Presence of Dependent Variables. The CHKPRE routine checked for the presence of dependent variable instances that had the specified attributes, that is, calculated from the named independent variables, and using the specified method and model. If the correct relationships were not found, new dependent instances were created and the instructions to calculate the values were placed in the code area.

On entry to CHKPRE the activation record is as shown in figure 64.

If one or more dependent variable instances already existed with the names the wagon designer provided, then this routine tested whether or not they had been calculated using the same method, model and independent variable instances. Firstly the model table name, databank, and column names were extracted from the activation record. The instance numbers for all variables except existing dependent instances were loaded from the activation record into local variables. The column number and stack pointer to the start of the list of existing dependent

instances were also loaded from the activation record into local variables.

Next the model table was searched for a row with the method equal to that specified in the activation record, and with the variable instance identities equal to the instance numbers in the activation record. The combinations of instances were searched for in the model table by attempting to retrieve rows, a row at a time, with constant method and variable instance numbers except for the existing dependent instances. These were varied through the combinations with the leading dependent variables varying more rapidly. When all instances for all the existing dependent variables had been used in the row selection criteria then the search for existing relationships in the model was completed.

If relationships were found, a message was displayed to the wagon designer. If all the dependent instances were unique then the stop flag was set and the routine exited. Otherwise the instance numbers were output for the relationship(s) found in the model table, before the flag was set and the routine exited.

If no relationships were found, new instances were created for all dependent variables with existing instances of the same name as specified by the wagon designer. The dependent number was found by searching the activation record for the column number of each of the dependent variables with existing instances. The name table, databank, and instance number column were extracted from the activation record for this dependent number. An instance was created using the INSID routine.

If there were no existing dependent instances or no relationships found for the existing instances and new instances had just been created

then the system proceeded with the calculation of the dependent variables. The stack top was changed to the top of the list of variable instance numbers by popping the column counter, current variable instance name, and existing dependent instances off the activation record. Routine MLOKUP was called to provide the instruction for the method. This instruction (the instruction to complete the creation of the dependent variable instances), and a return and a stop instruction were placed in the code area so they were the next instructions to be executed. The routine then exited.

If, on return to INSTRN, the stop or error flag was set the string storage locations were freed, the flag reset, and the return and stop instructions placed in the code area as the next instructions to be executed. Otherwise if the flag was not set the instruction counter was increased.

The structure chart is presented in figure 66.

(g) Calculation Instructions. Two instructions were implemented: one numbered the degrees of freedom, found the semi-bandwidth of the stiffness matrix, and then formed a banded stiffness matrix for a finite element model; the other formed the force vectors for a number of load cases (the right-hand side matrix) for a finite element analysis.

Both instructions extracted the variable instance numbers from the activation record. If an error occurred the instruction counter was increased and the value "one" was pushed on to the top of the stack (the calculation return code location) and the instruction exited. The assemble right-hand side and the form stiffness matrix instructions then called ASRHS and STIFF routines respectively.

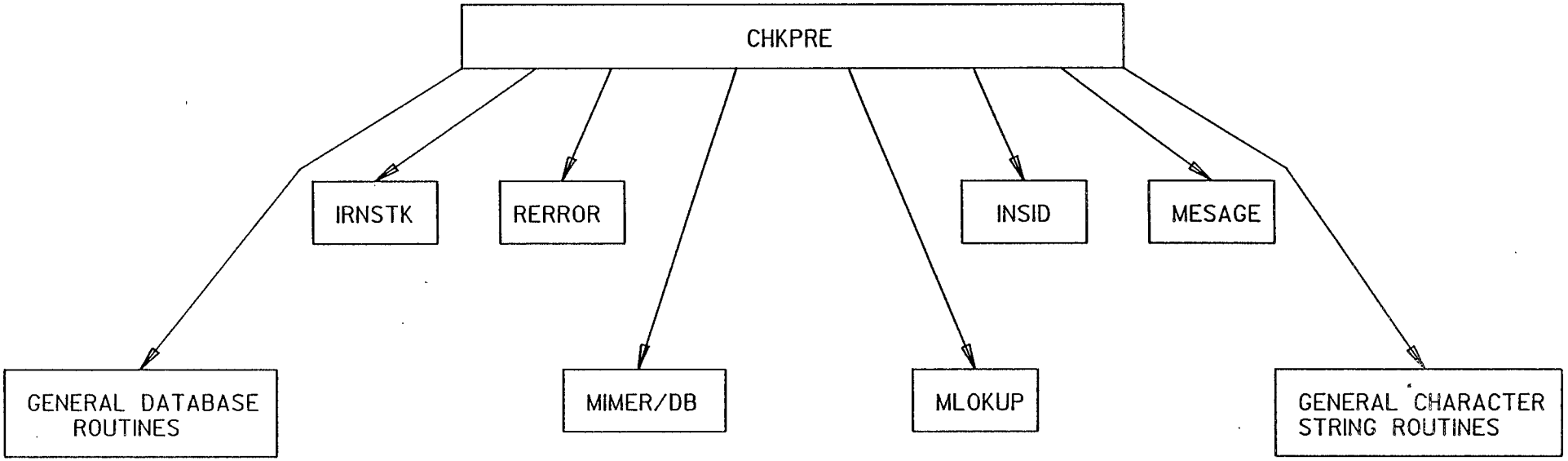


Figure 66. Structure Chart for the CHKPRE Routines

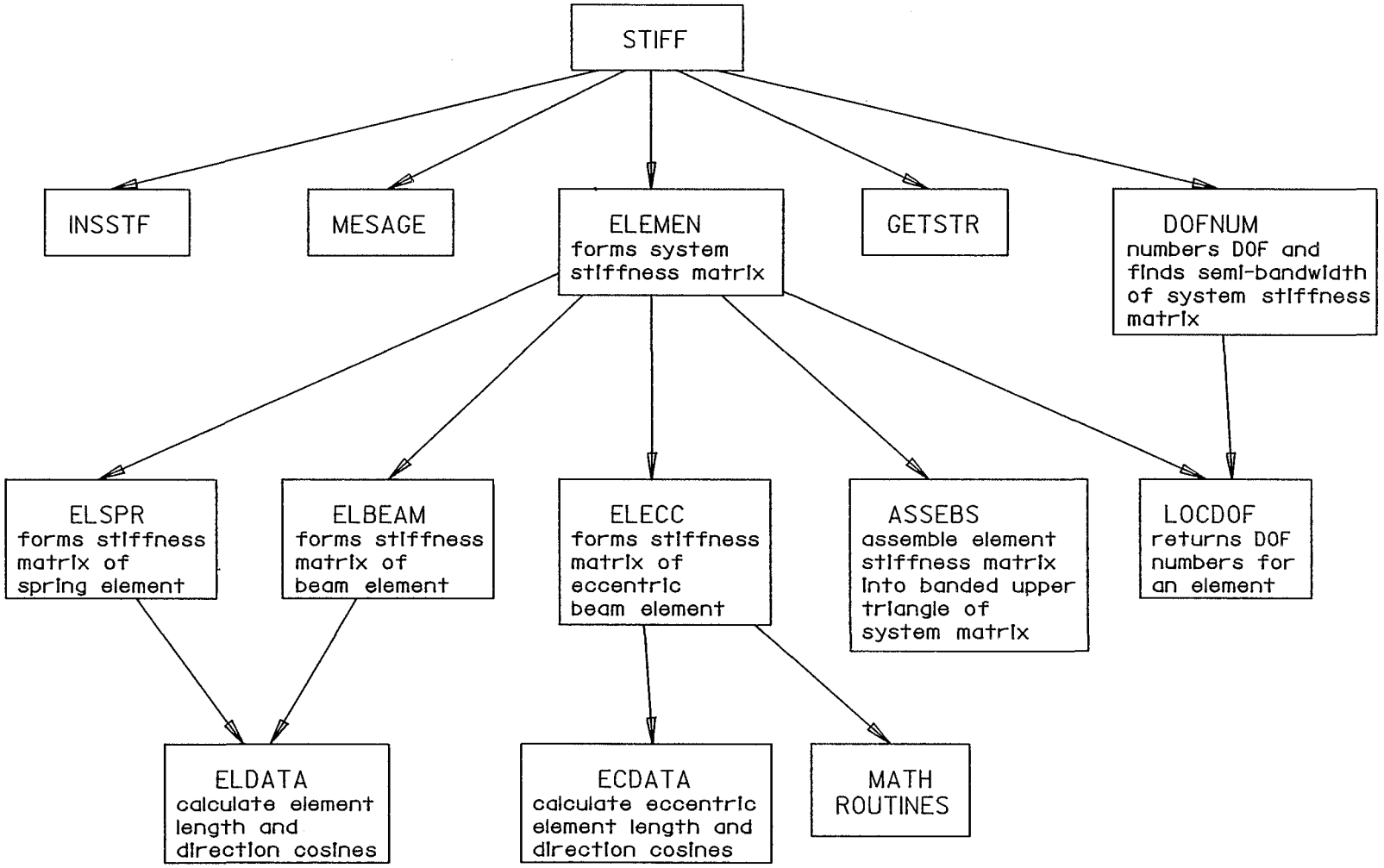


Figure 67. Structure Chart for STIFF Routines

The structure instance number and the stiffness matrix instance number were passed to the STIFF routine which retrieved the structure definition from the FEM databank, performed the calculations and then inserted the stiffness matrix into the databank (refer fig. 67).

The ASRHS routine was passed the structure, stiffness matrix, load set, and load matrix instance numbers. It used routines to retrieve the stiffness matrix, load definitions, element and node data. Another routine performed the assembly of the matrix and finally the load matrix was inserted into the FEM databank (refer fig. 68). Further details on the functions of the finite element routines can be found in most texts on the subject.

If an execution error occurred in these routines, the value "one" was pushed on to the top of the run-time stack and the instruction counter increased. Finally the instruction finished and INSTRN returned to EXECUT.

The activation record on completion of the calculation instructions was as shown in figure 69.

(h) Complete Creation of Dependent Variable Instance. The COMDBI routine completed the creation of dependent variable instance(s) by updating the attributes in the name table(s) and inserting a row into the model table.

If the calculation of values was successful, as indicated by the calculation return code on the run-time stack, then the updating proceeded. The name table was updated for each dependent variable. The name table, databank, and instance number column name were obtained from the activation record as were the dependent instance name and number. The instance name, source model, and instance creation date attributes

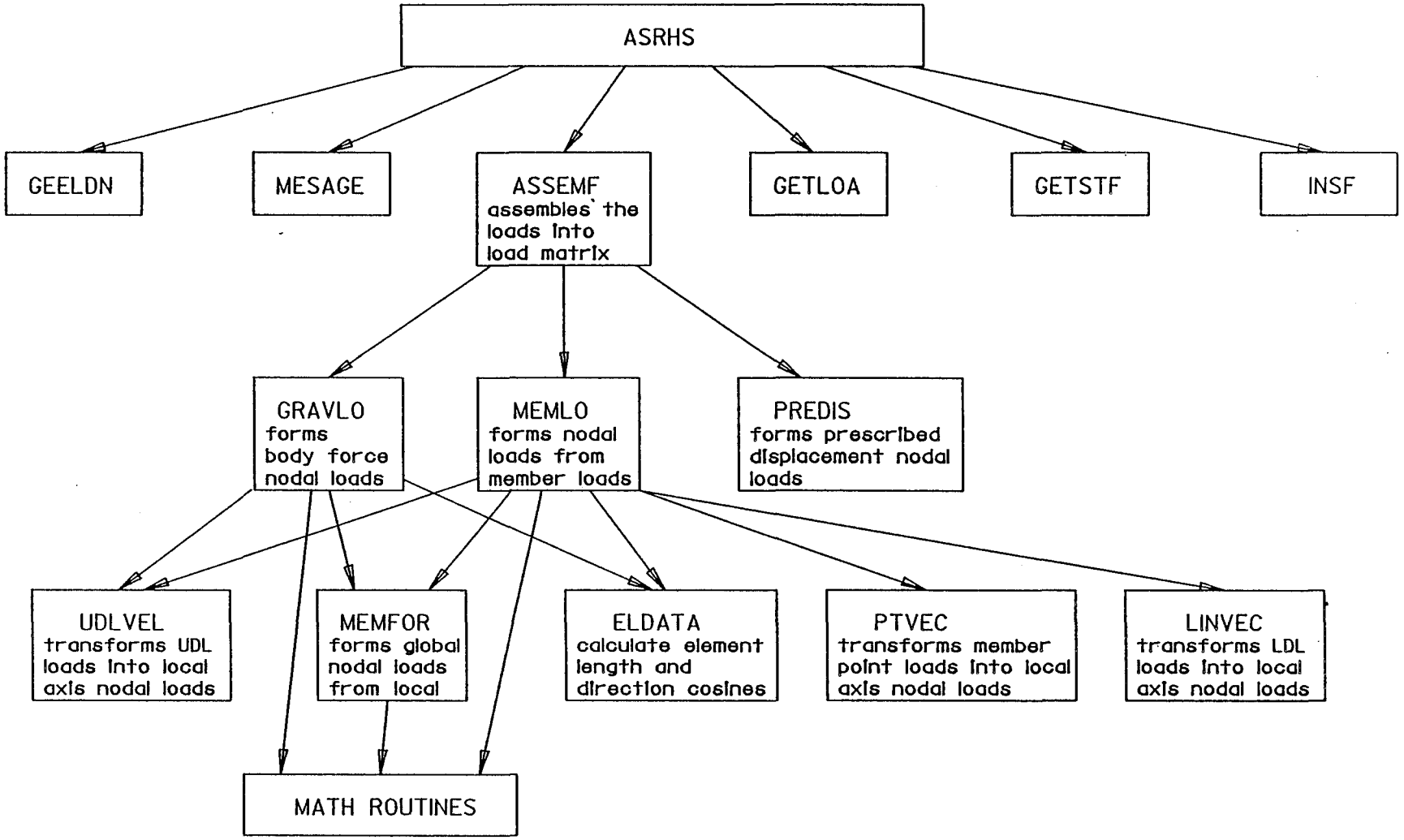


Figure 68. Structure Chart for ASRHS Routines

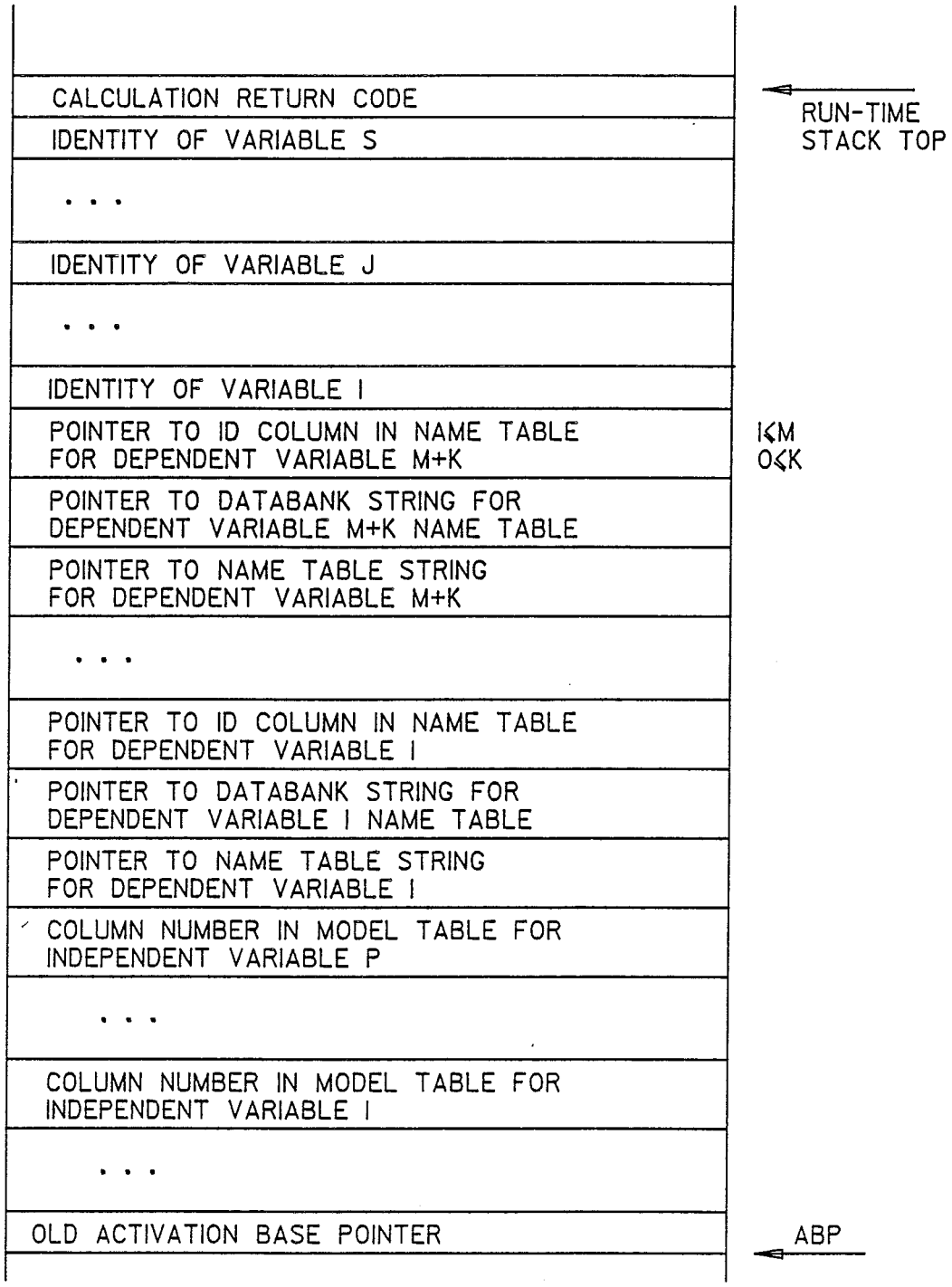


Figure 69. Activation Record on Exit from the Calculation Routines

were updated for the row with the instance number the same as the dependent instance. If the instance number was not found MIMCON handled the constraint violation.

Once the name tables had been updated, this routine inserted a row into the model table. The model table name, databank name, column names, the model, and the independent instance numbers were extracted from the activation record. A row was created in the model table consisting of the instance numbers and method. If no errors occurred the routine exited.

If an error had occurred during the calculation of the values or during the execution of this routine then the error was reported. Not implemented but part of the specification was to ensure integrity by using the MIMER transaction management for write operations on the database or alternatively deleting the partially created instances and all data in the database where the instance number (the entity key) appears in a relation (thereby ensuring referential integrity).

The structure chart for COMDBI appears in figure 70.

3. INTERPRETER RESPONSES

(1) Run-Time Errors

As the intermediate code was interpreted, conditions arose in which execution could continue or had to be rectified. When such conditions were detected a run-time error had occurred. The interpreter reported the error and terminated execution of the command if the error conditions could not be rectified.

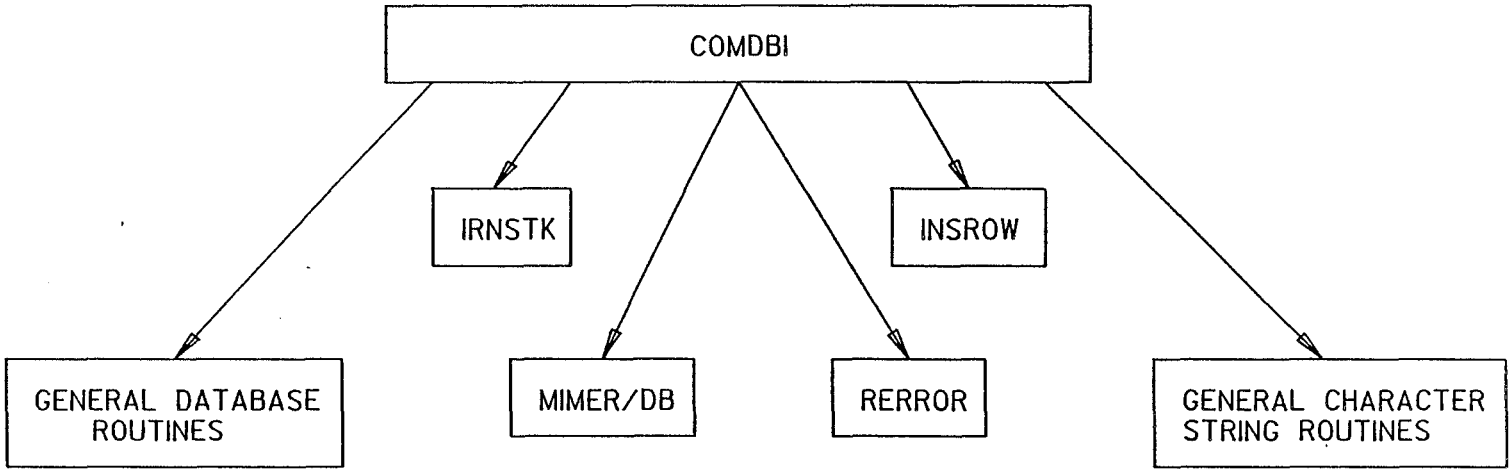


Figure 70. Structure Chart for the COMDBI Routines

Command or logic errors such as division by zero is one cause of errors. The wagon designer must fix the logic of the command for this type of error. A second cause of errors occurs when there is no more free space in the string area or run-time stack and more memory is being requested. For this type of error, the size of these areas must be increased and the interpreter has to be recompiled. The third cause of errors is invalid code generated by the interpreter, such as an illegal or unknown instruction is to be executed.

Appendix XII lists the error messages.

(2) Informative Messages and Requests

These responses to commands request more information in order to try and complete the command and also inform the wagon designer as to progress of the execution of the command. Appendix XII lists the informative messages and requests in the prototype design system.

4. REFERENCES

TREMBLAY, J.B. and SORENSON, P.G. (1982) An Implementation Guide to Compiler Writing. New York, McGraw-Hill. 259p.

CHAPTER XIX

PROTOTYPE INTEGRATED DESIGN SYSTEM SAMPLE SESSION

This chapter gives an example of a prototype design system session. Included with the source listing is a brief explanation of the session and some resource usage information.

1. THE SESSION

The session (refer fig. 71) begins with the operating system command to run the prototype system. The ">" sign indicates the system is ready for a command; lines without the ">" are system responses. The first command is an assignment. The second line contains an assignment and a write command. The third command has incorrect syntax and the error message displays valid terminals (keywords, constants or identifiers) that could have appeared. The fourth command illustrates character string variables. The next command has a syntax error and possible valid terminals are displayed again.

The solve command is given to calculate the RHS entity using the ASSEMBLERHS model. The wagon designer is prompted for a method and he accepts the default which is RHSASSEMBLER. He is then prompted for the identity of the load and structure neither of which have default values at this stage. The system finds that there are two structure instances with the given name, and asks the wagon designer to select one. He is then prompted for stiffness name, to which he responds by entering the query language initiation command.

```
RUN IDS/W

#RUNNING 2966

>A=3*4-32.5*4;

>SECOND=COS (360/3.142*45); WRITE 'A*SECOND =', A*SECOND;

A*SECOND = 97.517437366

>ANERROR=4***1;

           1

? ERROR(1): EXPECTING SQRT

            OR SIN

            OR EXP

            OR COS

            OR ATAN

            OR ALOG10

            OR ALOG

            OR ABS

            OR NOT

            OR -

            OR (

            OR TRUE

            OR FALSE

            OR LITERAL

            OR NUMBER

            OR SIMPLE IDENTIFIER

            BUT FOUND *

>ASTRING='HELLO WORLD'; WRITE ASTRING;

HELLO WORLD
```

Figure 71. A Sample Prototype Design System Session

```
>ANERROR=12 A;
            1
? ERROR(1):EXPECTING *
           OR /
           OR AND
           OR +
           OR -
           OR OR
           OR GT
           OR LT
           OR EQ
           OR GE
           OR LE
           OR NE
           OR )
           OR END OF COMMAND
           OR ,
           BUT FOUND SIMPLE IDENTIFIER

>SOLVE ASSEMBLERHS FOR RHS;
USING  :  ( RHSASSEMBLER      )?
>;
LOAD          : (             )?
>'WORST CASES';
STRUCTURE  : (          )?
>'BTK VERSION 3';
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
     NAME              SOURCE MODEL                    DATE
-------------------------------------------------------------
1,   BTK VERSION 3     DESIGNER INPUT                  300984
2,   BTK VERSION 3     DESIGNER INPUT                  310984
'BTK VERSION 3' DOES NOT SPECIFY UNIQUE STRUCTURE INSTANCE
- SELECT 1-2
>1;
STIFFNESS     : (               )?
>QL;

                        M I M E R / Q L
                        Version 3.1.4

Username: MECH090
Password:
QL> INCLUDE DATADICT;
QL>GET TABDICT.* WHERE TABDICT.DESCRIPT CO 'Stiffness';

---------------------------------------
DATABANK   TABLE     DESCRIPT
---------------------------------------
FEM        SMATRIX   Stiffness matrices
           1 row(s) found
QL>GET TABDICT.TABLE, TABDICT.DESCRIPT
+> WHERE TABDICT.DATABANK EQ 'FEM';
```

Figure 71. A Sample Prototype Design System Session (cont'd)

TABLE	DESCRIPT
ACCLN	Acceleration values
ASSEMFOR	Assembling RHS relationships
BEAMEL	Beam elements
BEAMSTRE	Beam stressing properties
BODYLOAD	Body load definitions
CONSTRDF	Constrained degrees of freedom
DELSET	Displacement set names
DISPLACE	Translational displacement values
DSETDEF	Displacement set definitions
ECCEL	Eccentric beam elements
ELEMENTS	Definition of elements
ELGROUP	Elements in element groups
ELSET	Name of element groups
FACTLOAD	Load factors
FORCES	Force values
FSET	Names of assembled RHS
FSETDEF	Assembled RHS matrices
LINEARML	Linear distributed loads
LOADCASE	Loadcase names
LOADGROU	Definition of sets of structural loads
LOADSET	Names of sets of structural loads
MEMBLOAD	Member loads
MOMENTS	Moment values
NODELOAD	Nodal loads

Figure 71. A Sample Prototype Design System Session (cont'd)

```
  NONSTD      Properties of non standard sections
  POSITION    Positions in global axis system
  PRESDISP    Prescribed degrees of freedom
  PRESDOF     Prescribed displacements definitions
  PTML        Point loads on members
  ROTATION    Rotation values
  SDEF        Names of stiffness matrices
  SDOFNO      Numbering of degrees of freedom
  SMATRIX     Stiffness matrices
  SPRGPROP    Spring properties
  SPRINGEL    Spring elements
  STATSTR     Loaded static structure relationships
  STIFFEN     Structure to stiffness matrix relationships
  STRNODES    Structural nodes
  STRUCDEF    Elements in structures
  STRUCTUR    Structure names
  UDLOAD      Uniform distributed load values
  UDML        Uniform distributed member loads

                          42 row(s) found

QL>INCLUDE FEM(SDEF);

QL>DESCRIBE TABLE SDEF;

      SDEF   1 SMATRIX  *              I    2
             2 SEMIBAND                I    2
             3 NDOF                    I    2
             4 TOTALEQ                 I    2
             5 NAME                    C    16  *Indexed*
             6 MODEL                   C    16  *Indexed*
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
             7 DATE                          C    6

QL>DESCRIBE TABLE COLDICT;

  COLDICT    1 DATABANK *                    C    8

             2 TABLE    *                    C    8

             3 COLUMN   *                    C    8

             4 KEY                           C    1

             5 DESCRIPT                      C    45

             6 UNITS                         C    10

QL>GET COLDICT.COLUMN,COLDICT.DESCRIPT WHERE COLDICT.TABLE EQ 'SDEF';

------------------------------------------------------------------

       COLUMN                DESCRIPT

------------------------------------------------------------------

      NAME          Stiffness matrix name

      NDOF          Number of degrees of freedom

      SEMIBAND      Semibandwidth of stiffness matrix

      SMATRIX       Stiffness matrix identity

      TOTALEQ       Total number of equations (incl. constrained)

                    5 row(s) found

QL>GET SDEF.*;

------------------------------------------------------------------

SMATRIX    SEMIBAND    NDOF    TOTALEQ    NAME          MODEL     DATE

------------------------------------------------------------------

   1         78          216       216    BTK VERSION 3  STIFFEN  012885

             1 row(s) found

QL>EXIT;
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
                    M I M E R  /  Q L

                Session time   :   44 min

                Control words  :  1088

                Page frames    :   30

                Page requests  :  1054

                Page faults    :   30

                    M I M E R  /  D B

>STIFFNESS='BTK TRIAL';

>SOLVE STIFFEN FOR STIFFNESS;

USING  :  (  BANDEDSTIFFENER  )?

>;

STRUCTURE    :  (              )?

>'BTK VERION 3';

'BTK VERION 3 '      STRUCTURE NOT PRESENT

   - SUSPENDED EXECUTION.

>QUIT;

SOLVE  :  ASSEMBLERHS

STIFFNESS    :   (BTK TRIAL    )?

>SOLVE STIFFEN FOR 'BTK TRIAL' STIFFNESS

+> WITH 'BTK VERSION 3' STRUCTURE

+> USING BANDEDSTIFFENER;
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
        NAME                    SOURCE MODEL                    DATE
-------------------------------------------------------------------

1,    BTK VERSION 3          DESIGNER INPUT              300984

2,    BTK VERSION 3          DESIGNER INPUT              310984

 'BTK VERSION 3'  DOES NOT SPECIFY UNIQUE STRUCTURE INSTANCE

  - SELECT 1-2

>1;


STRUCTURE     1 SUCCESSFULLY RETRIEVED FROM DATABASE...


EL.NO   1      NMAX,NMIN    78    1     S-B'WIDTH      78

EL.NO   2      NMAX,NMIN   102   73     S-B'WIDTH      30

EL.NO   3      NMAX,NMIN   126   97     S-B'WIDTH      30

EL.NO   4      NMAX,NMIN   198  121     S-B'WIDTH      78

EL.NO   5      NMAX,NMIN    30    7     S-B'WIDTH      24

EL.NO   6      NMAX,NMIN    60   25     S-B'WIDTH      36

EL.NO   7      NMAX,NMIN    84   55     S-B'WIDTH      30

EL.NO   8      NMAX,NMIN   108   79     S-B'WIDTH      30

EL.NO   9      NMAX,NMIN   132  103     S-B'WIDTH      30

EL.NO  10      NMAX,NMIN   150  127     S-B'WIDTH      24

EL.NO  11      NMAX,NMIN   192  145     S-B'WIDTH      48

EL.NO  12      NMAX,NMIN   204  187     S-B'WIDTH      18

EL.NO  13      NMAX,NMIN    36   13     S-B'WIDTH      24

EL.NO  14      NMAX,NMIN    42   31     S-B'WIDTH      12

EL.NO  15      NMAX,NMIN    48   37     S-B'WIDTH      12

EL.NO  16      NMAX,NMIN    66   43     S-B'WIDTH      24
```

Figure 71. A Sample Prototype Design System Session (cont'd)

EL.NO	17	NMAX,NMIN	90	61	S-B'WIDTH	30
EL.NO	18	NMAX,NMIN	114	85	S-B'WIDTH	30
EL.NO	19	NMAX,NMIN	138	109	S-B'WIDTH	30
EL.NO	20	NMAX,NMIN	156	133	S-B'WIDTH	24
EL.NO	21	NMAX,NMIN	168	151	S-B'WIDTH	18
EL.NO	22	NMAX,NMIN	180	163	S-B'WIDTH	18
EL.NO	23	NMAX,NMIN	186	175	S-B'WIDTH	12
EL.NO	24	NMAX,NMIN	210	181	S-B'WIDTH	30
EL.NO	25	NMAX,NMIN	12	1	S-B'WIDTH	12
EL.NO	26	NMAX,NMIN	18	7	S-B'WIDTH	12
EL.NO	27	NMAX,NMIN	24	13	S-B'WIDTH	12
EL.NO	28	NMAX,NMIN	54	43	S-B'WIDTH	12
EL.NO	29	NMAX,NMIN	66	55	S-B'WIDTH	12
EL.NO	30	NMAX,NMIN	72	61	S-B'WIDTH	12
EL.NO	31	NMAX,NMIN	84	73	S-B'WIDTH	12
EL.NO	32	NMAX,NMIN	90	79	S-B'WIDTH	12
EL.NO	33	NMAX,NMIN	96	85	S-B'WIDTH	12
EL.NO	34	NMAX,NMIN	108	97	S-B'WIDTH	12
EL.NO	35	NMAX,NMIN	114	103	S-B'WIDTH	12
EL.NO	36	NMAX,NMIN	120	109	S-B'WIDTH	12
EL.NO	37	NMAX,NMIN	108	97	S-B'WIDTH	12
EL.NO	38	NMAX,NMIN	114	103	S-B'WIDTH	12
EL.NO	39	NMAX,NMIN	120	109	S-B'WIDTH	12
EL.NO	40	NMAX,NMIN	132	121	S-B'WIDTH	12
EL.NO	41	NMAX,NMIN	138	127	S-B'WIDTH	12
EL.NO	42	NMAX,NMIN	144	133	S-B'WIDTH	12
EL.NO	43	NMAX,NMIN	156	145	S-B'WIDTH	12

Figure 71. A Sample Prototype Design System Session (cont'd)

```
EL.NO  44     NMAX,NMIN   162  151    S-B'WIDTH     12
EL.NO  45     NMAX,NMIN   174  163    S-B'WIDTH     12
EL.NO  46     NMAX,NMIN   204  193    S-B'WIDTH     12
EL.NO  47     NMAX,NMIN   210  199    S-B'WIDTH     12
EL.NO  48     NMAX,NMIN   216  205    S-B'WIDTH     12
EL.NO  49     NMAX,NMIN    42    7    S-B'WIDTH     36
EL.NO  50     NMAX,NMIN   204  175    S-B'WIDTH     30
ASSEMBLED STIFFNESS MATRIX S-BANDWIDTH IS  78
STIFFNESS 2 SUCCESSFULLY ASSEMBLED FOR STRUCTURE  1...
STIFFNESS 2 SUCCESSFULLY INSERTED IN DATABASE ...
SOLVE : ASSEMBLERHS
STIFFNESS    : ( BTK TRIAL    )?
>QL;

                    M I M E R  /  Q L

                      Version 3.1.4

Username:  MECH090
Password:
QL>INCLUDE  FEM(SMATRIX,SDEF,STIFFEN,SDOFNO);
QL>INCLUDE  DATADICT;
QL>GET  COLDICT.COLUMN,COLDICT.DESCRIPT  WHERE
+> COLDICT.TABLE EQ 'SMATRIX';
```

Figure 71. A Sample Prototype Design System Session (cont'd)

COLUMN	DESCRIPT
IDOF	I degree of freedom
JDOF	J degree of freedom
SMATRIX	Stiffness matrix identity
SVALUE	Value of stiffness matrix element

4 row(s) found

```
QL>SET LC='20';
QL>GET SMATRIX.* WHERE SMATRIX.IDOF GT '200';
```

SMATRIX	IDOF	JDOF	SVALUE
1	201	201	4226427511.45
1	201	202	8816008.8586
1	201	203	11611427.5355
1	201	204	0
1	201	205	0
1	201	206	0
1	201	207	-18842841 40.98
1	201	208	0
1	201	209	0
1	201	210	0
1	202	202	133597324.908
1	202	203	0
1	202	204	173255.922463

Figure 71. A Sample Prototype Design System Session (cont'd)

```
   1              202              205                  0
   1              202              206        202119680.184
   1              202              207                  0
Continue ...? S
***Output interrupted ***
QL>GET STIFFEN.*;

-------------------------------------------------------------------

             STRUCTUR          SMATRIX              METHOD
-------------------------------------------------------------------

                 1                 1          BANDEDSTIFFENER
                 1                 2          BANDEDSTIFFENER
          2 row(s) found
QL>GET COLDICT.COLUMN,COLDICT.DESCRIPT WHERE COLDICT.TABLE EQ SDOFNO';

-------------------------------------------------------------

     COLUMN              DESCRIPT
-------------------------------------------------------------

  DX                Degree of freedom number:X displacement
  DY                Degree of freedom number:Y displacement
  DZ                Degree of freedom number:Z displacement
  RX                Degree of freedom number:X rotation
  RY                Degree of freedom number:Y rotation
  RZ                Degree of freedom number:Z rotation
  SMATRIX           Stiffness matrix identity
  STRNODNO          Structural node number
     8 row(s) found
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
QL>GET SDOFNO.* WHERE SDOFNO.SMATRIX EQ '2' AND SDOFNO.STRNODNO LT '25';

-------------------------------------------------------------
SMATRIX  STRNNO      DX     DY     DZ     RX     RY     RZ
-------------------------------------------------------------
   2        1         1      2      3      4      5      6
   2        2         7      8      9     10     11     12
   2        3        13     14     15     16     17     18
   2        4        19     20  -21.4  -22.4  -23.4     24
   2        5        25  -26.4     27     28     29     30
Continue ...? S
*** Output interrupted ***
QL>EXIT;

                    M I M E R  /  Q L
                Session time   :    7 min
                Control words  :    1105
                Page frames    :    30
                Page requests  :    681
                Page faults    27
                    M I M E R  /  D B
>RETURN;
SOLVE  :  ASSEMBLERHS
STIFFNESS   :   ( BTK TRIAL       )?
>SOLVE STIFFEN FOR 'BTK TRIAL' STIFFNESS
+>WITH 'BTK VERSION 3' STRUCTURE
+>USING BANDEDSTIFFENER;
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
    NAME                SOURCE MODEL                DATE
-------------------------------------------------------

1,  BTK VERSION 3       DESIGNER INPUT        300984
2,  BTK VERSION 3       DESIGNER INPUT        310984
'BTK VERSION 3 ' DOES NOT SPECIFY UNIQUE STRUCTURE INSTANCE
-SELECT 1-2
>1;
ENTITY(S) EXISTS WITH THESE ATTRIBUTES AND RELATIONSHIPS.
SOLVE   : ASSEMBLERHS
STIFFNESS   : ( BTK TRIAL    )?
>;
RHS      : (            )?
>'BTK EXPERIMENT';
STRUCTURE       1 SUCCESSFULLY RETRIEVED FROM DATABASE...
LOAD            1 SUCCESSFULLY RETRIEVED FROM DATABASE...
STIFFNESS       2 SUCCESSFULLY RETRIEVED FROM DATABASE...
WARNING: NODE  5, DOF NO.  -26 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
         NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 5, DOF NO. -26 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
         NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 10, DOF NO.  -56 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
         NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 10, DOF NO.  -56 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
         NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 25, DOF NO.  -146 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
```

Figure 71. A Sample Prototype Design System Session (cont'd)

```
        NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 25, DOF NO.  -146 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
        NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 32, DOF NO.  -188 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
        NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 32, DOF NO.  -188 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
        NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 31, DOF NO.  -181 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
        NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
WARNING: NODE 31, DOF NO.  -181 IS A PRESCRIBED DISPLACEMENT DOF AND HAS
        NOT HAD ITS LOAD APPLIED IN LOAD CASE 1
RHS 1 SUCCESSFULLY ASSEMBLED FOR STRUCTURE 1, STIFFNESS 2, AND LOAD 1...
FSET 1 SUCCESSFULLY INSERTED IN DATABASE...
>SOLVE STIFFEN FOR STRUCTURE;
? :LINE(   24):NO DEFAULT METHOD FOR SOLVING STIFFEN
FOR THIS ENTITY(S).
>SOLVE STIFFEN FOR STRUCTURE USING BANDEDSTIFFENER;
?: LINE(   25):THIS METHOD CAN NOT BE USED FOR CREATING STRUCTURE IN
THIS MODEL.
>END
```

Figure 71. A Sample Prototype Design System Session (cont'd)

In the query language subsystem the wagon designer specifies that he wishes to work with the Data Dictionary databank. He retrieves the row of the table dictionary where the description contains the character string "stiffness". Then the table names and description fields are retrieved for all the tables in the FEM databank. The SDEF table in the FEM databank is included in the work area, and the definition of both the SDEF and COLDICT tables are listed. The column names and descriptions are retrieved for the SDEF table. The contents of the SDEF table are then retrieved, before MIMER/QL is exited.

The variable STIFFNESS is assigned a name and then the command to solve STIFFEN model for STIFFNESS is entered. The wagon designer typed BTK VERION 3 as the STRUCTURE identity (perhaps mistakenly) and after the system responds that this STRUCTURE is not stored, the wagon designer quits that solve command. After the quit command the system responds with the status of the previously suspended solve command (now with the default value for the stiffness identifier). The solve for stiffness command is entered in full, and execution begins after selecting the particular structure required. Various messages are output indicating the progress of the command.

The query language subsystem is re-entered and various retrievals are performed on the finite element data and Data Dictionary.

After exiting the query language subsystem, the same solve command is keyed in but this time the system responds that a stiffness matrix already exists with the specified attributes and relationships.

The suspended solve command is then executed successfully. Finally two attempts are made to calculate a structure using the STIFFEN model, both rejected by the system.

2. RESOURCE USAGE

(1) Timings

Timings were recorded for the prototype design system commands by using the Burroughs FORTRAN TIME intrinsic. They showed that the simple assignment and write commands used approximately 0.1 seconds CPU and 0.1 seconds I/O time. Response times were well under a second on a lightly loaded system.

The various routines of the solve command other than the method instructions (for example, retrieve an independent variable identity) took at most 1 second of CPU and 0.3 to 0.8 seconds of I/O. Retrieval and in particular insertion of the finite element data took significantly longer than the "in-core" solution steps. For example, for the model discussed in chapter XV (50 beam elements, 36 nodes, 6 beam property instances, 32 constrained DOF, 4 load cases with 7 nodal loads and 2 UDL's defined on a total of 6 elements), retrieval of the data from the database into the programs data arrays took 2.2 times more CPU (4.5 seconds), 3.3 times (0.8 seconds) more I/O and 1.6 times the elapsed time (8.7 seconds) than the numbering of the degrees of freedom and the formation of the stiffness matrix. The insertion of the stiffness matrix and degree of freedom data into the database took 19.6 times as much CPU, 2.1 times the I/O and 8.7 times the elapsed time (47.7 seconds). Retrieval of the stiffness matrix and degree of freedom numbering took 1.8 times the CPU usage of the computation step, 0.6 times the I/O and 1.1 times the elapsed time. The whole solve command took 51.6 seconds CPU, 4.6 seconds I/O and 84 seconds elapsed time. The

assembly of the right-hand side matrix had similar ratios. No extra effort was spent in attempting to optimise the code, although some routines such as INSF were quite straight forward.

(2) Storage Requirements

There were significant storage overheads as well. The data defining the structure (plus some empty tables) took approximately 120 KBytes in the database, but the input file size for the sequential file version of the finite element program used at most 12.8 Bytes. Approximately 110 kBytes were used to store approximately 53 kBytes of row data in the WAGON databank. The increase in required space indicates that indexing and used-to-total-allocated space efficiency (claimed to be 83% on average) have a significant effect. (Some tables required a page of disk storage for the B*-tree index nodes to a page of disk storage for the data nodes, although this is dependent on the number of rows, row length and key length.)

CHAPTER XX

COMMENTS AND CONCLUSION

In this chapter, comments on the implementation exercise are presented, views are expressed on further development and, finally, the conclusion of the thesis is presented.

1. THE FEASIBILITY STUDY

(1) Technical Feasibility

The prototype integrated design system has demonstrated a way to provide wagon designers with an integrated environment. A means of relating design data has been successfully applied to two analysis procedures.

Response times for simple commands was subsecond. Response times for the solve command, particularly those parts concerned with updating the database, could benefit from improvement but were for the most part still acceptable considering the processing performed.

Total program code and data size was large but as only a subset is required at a time (as directed by the wagon designer) memory requirements should pose no problems for an expanded system (refer sect. 2(2) for a suggested implementation improvement).

The Geometry and Wagon databanks together occupied 217 kBytes of storage space. A complete description of a wagon would occupy an estimated fifty times this space. If one hundred (completely different) wagon designs were stored then disk space requirements would be of the

order of 1000 MBytes. When an allowance is added for catalogue data, the data dictionary and the temporary project data the total disk space requirements are still well within the capacity of most "mini-computers".

(2) Economic Feasibility

(a) Costs. If it is assumed that maintenance charges, overhead allocations, and depreciation were included in the University of Canterbury's estimate of a realistic commercial charge out rate for Burroughs 6900 CPU time ($5.10 per minute) then it would have cost under $5.00 to assemble the stiffness matrix for the example in chapter XIX using the prototype scheme (and less than $5.00 to assemble the right-hand side matrix).

The size of the sequential file storing the input data for the sequential file based version of the finite element program was 12.8 kBytes, approximately 107 kBytes less than the same data stored in the DBMS. If disk storage costs using modern Winchester technology are conservatively estimated at $150 per MByte then the database overhead costs $16.05. (Storage on magnetic tape is considerably cheaper at approximately 50 cents per MByte.)

Several more man-years of programming are required before the design system will be of sufficient scope to be of significant value. The other incremental start-up charges are made up of the purchase prices of the DBMS (at the time MIMER cost approximately $30,000 on a PRIME 750), screen formatter/menu generator and software development tools. (By incremental charges, the author means those in addition to application package and hardware initial costs as these form the bulk of

the startup costs of the non-integrated computerised solution as well.)

(b) Benefits. Computer systems have attributes that have given proven benefits in areas such as design analysis, draughting graphics, and in large commercial databases and show promising potential in areas such as expert systems. For example, in wagon design, Elkins and Eickhoff (refer chapt.V, sect. 4) have reported use of their computer based nonlinear curving model can lead to a reduction by a factor of 10 in flange and rail wear. This same model can calculate drag, fuel consumption, and stressing forces for 0.25 to 0.5 seconds of IBM 370-168 processor time for each case studied. Finite element programs permit much more extensive study than would be normally considered feasible by manual analysis. Determination of fatigue life of a component under random cyclic loading can be satisfactorily resolved only by the use of a computer. Chapter III, section 5 contains reference to a study in which was reported a reduction in weight of a production underframe of 35% using a combination of the finite element method and structural optimisation.

The examples above may be the best in terms of benefits of computer based analysis, but positive returns (perhaps not as impressive) are also obtained on a much wider range of computer based analyses.

Then, are there proven benefits for the scheme which integrates individual computer based analyses and design procedures (for example, the works of the researchers reported in chapters II-VIII and the procedures used by NZR wagon designs also reported in these same chapters)?

The answer from a quantitative point of view is no. The lack of quantified benefits should be addressed as soon as at least two

(preferably more) design application programs have been written into the prototype system. Then use of the prototype system in the Design Office environment with wagon designers as users will enable testing to provide quantified benefits (for example, estimates of productivity improvements) and also help to establish whether the authors ranking of objectives is the same as NZR. (Re-ranking could follow.) Following quantification of benefits a fuller comparison with the non-integrated approach would be possible.

However the lack of quantitative data on benefits does not prevent a qualitative assessment. A number of features from those identified in the system specification were implemented in the prototype system. These features were identified as providing benefits in the areas outlined in chapter I, section 2(1)(b).

An interactive command language with calculator and design procedure/analysis functions was implemented and interfaced with a commercial database query language which offered complementary functions of data manipulation and definition. The command language had syntax similar to the query language but used wagon designer constructs. It prompted for variables and data not supplied in the command line. Commands that specified what to do to a set of inputs in a single command were possible.

All data could be either calculated using computer based methods or input by the wagon designer. The wagon designer decided what data was used as input, what variables were calculated using a method which was also specified by the wagon designer (within the constraints imposed by the variables and solution methods programmed). Default methods and values were displayed when required.

The database system provided fast and easy means to retrieve or store all data. Routine queries could be answered in a matter of seconds instead of minutes as is the case for the manual system. The query language provided a means to handle unanticipated requests. Access control was also a feature of the database. The database acted as a common repository of data sharing and communicating the data in a reliable and organised manner to different applications.

The relationships between data were stored documenting how values were arrived at and providing a history of the design. These relationships ensure integrity of updates to variable values. Any number of values for a variable were permitted, each value's relationship with other variable values being maintained. Thus trial solutions and experimentation were facilitated. The system informed the wagon designer what input data was not present in the database, suspending execution of the command and permitting calculation of the missing data, quitting of the suspended command, or other commands to be entered. Suspended commands could be resumed (in order of suspension) when the wagon designer wished. If the output data was already present in the database, the calculation commands informed the wagon designer and finished without recalculating the data.

Physical data independence and some degree of logical data independence was present in the database. The table driven command interpreter and system portability will also ease expansion and modification.

Redundancy in both the database and programmed functions was controlled.

An existing application program was integrated with the prototype

system without any rewriting of the application code.

It is these features which will remove the need to re-enter data when moving from one analysis program to another, that provide a clear, consistent, easy to use human interface, that improve design management control and co-ordination, that provide flexible means of accessing existing design properties and analysis results, that provide savings in system development costs and storage requirements, and so on.

These are the productivity, product quality and rationalisation improvements introduced in chapter I, section 2(1)(b) that are attributable to an integrated wagon design system.

(c) Cost Benefit Balance. Although there are significant development costs and machine overheads in an integrated approach compared with the other approaches (refer section (a)), these would seem to be small in comparison with potential savings in labour costs (with the minimum rate of $10 per hour). For instance, if it takes a draughtsman up to a year to attain proficiency in one computer-aided draughting system then rationalisation of languages and functions between other applications and draughting could show considerable savings in learning time.

Similarly when finite element models are created using the design geometry already within the database significant productivity gains are possible.

It should be noted that although the prototype system used considerably more storage space than the file based program, the addition of more applications, more projects (so that all tables contain data), and inclusion of more trial analyses for the one project should show the advantages of an integrated system in a better light. It is

also possible that with information co-ordination, ease of use, shared data, and so on, the CPU usage for a design project for the integrated scheme would decrease relative to the file based system.

With the staged development possible with a system such as proposed, it is possible (and advisable) to offset development costs against benefits (thereby providing a better rate of return) by operating the system as soon as applications are implemented. However, because the number of information flows increase at a greater than linear rate with application additions, benefits will be larger the wider the scope of the system.

(3) Operational Feasibility

The increased use of computers within the Design Office has raised the general level of computing knowledge (particularly noticeable with the introduction of a proprietary draughting/design system). The benefits in an integrated system would, in the opinion of the author, ensure the ready acceptance of such a scheme.

(4) The Preferred Solution

No general system framework suitable for development in computer-aided wagon design has been found that satisfies the requirements in chapt. X to XIII. Almost all commercial systems centre around large proprietary programs with strengths in either analysis or draughting. Generally these systems are deficient in their range of applications and provide little or no means of integrating such site dependent design procedures as wagon brake performance calculations. Generally they do not offer a means of storing the relationships between design data in a

structured and shared database. By storing the relationships between design data, integrity can be maintained even amongst the data for multiple design trials. They also lack in direction and control flexibility, including the ability to suspend an analysis to perform another task.

This thesis has shown that an integrated design scheme which includes many of the features lacking in other systems is feasible. As a general strategy it is preferred over the alternative strategies of independent file based programs and of interfacing through data files.

The author believes that the benefits are such that an integrated scheme will eventually be installed for NZR wagon design whether a conscious decision is taken now or not.

Other than the "do nothing" option, management can decide either to develop such a system internally using the subsystems available as building blocks or they can decide to wait for (and possibly encourage) commercial development of the integrated scheme. This later course involves the lesser risk (and cost) but precludes gain of potential competitive advantage.

If internal development is the decision, the prototype system and functional specification reported in this thesis can act as a starting point, alternatively the specification can be used in the selection of commercial systems.

2. COMMENTS ON THE PROTOTYPE IMPLEMENTATION

This section offers some criticisms on what was attempted in the implementation of the prototype integrated design system.

(1) The Overall Structure

The architecture of a user environment, command interpreter, run-time interpreter, and database with their respective interfaces (the command language; the code stack and data areas; and the data sublanguage) would seem quite sound. This architecture is similar to other modern systems. A modern DBMS, encoding language, operating system, and screen formatter/menu generator are a necessary basis for an efficient, flexible integrated design system.

(2) The Run-Time Interpreter

Execution efficiency improvements could be made at the expense of storage efficiency if the variable names were stored in the activation record as well as variable identifying numbers rather than retrieving them from the database.

The run-time interpreter itself should not have to know about the data arrays in the instruction routines. This aspect would have been better implemented if these instruction routines (such as the finite element routines) were self contained within an executable file which was initiated as a separate task or library program. Then the instruction routine would be passed variable instance identifiers and communicate messages back to the wagon designer. Upon completion the task would inform the run-time interpreter of completion status and then terminate.

In an environment where deletions are possible, inserting instances with identities equal to the number of rows in the table will cause relationship problems unless the referential integrity rule is enforced

and all references to the variable instance are deleted at the time the instance itself is deleted. Otherwise a time-stamp or count of all rows inserted for all time will be required for instance identifying.

Inserting new identities equal to the row count will also not necessarily give unique identities as intermediate identities could have been deleted before the next insertion. Again other mechanisms are required such as a time-stamp, running count of all instances inserted, or one greater than the highest identifier present, or just the lowest identifier not present.

The solve routines did not check that a supplied independent variable was really a dependent variable in the model method combination. Without reference to the supplied independent variable the routines retrieved independent variables from the model and method definitions. Thus the wagon designer had to supply the instance name twice if it was supplied in error as an independent variable. The separation between supplied independent variable instance names and instance names not supplied for dependent variables is desirable because the interpreter can not determine whether the dependent variable instance should or should not be the same as that supplied as an independent variable instance.

Generation of code in the run-time interpreter was ad hoc and could have been done using the parsing and code generation routines of the command interpreter. The handling of the suspend condition between the run-time interpreter and the command interpreter could also have been improved, particularly in the area of type checking.

(3) The Database

The ability to be able to directly load analysis program data arrays where the program is proprietary depends on the availability of the source code or routines that will input the data. If neither is available then the program will probably have to be interfaced by means of an intermediate file as most commercial programs can take input from an operating system file.

This approach may indeed be faster at loading the database than run-time data structure to database loading, that is, writing a matrix out to an operating system file and then performing a bulk load into the database.

Flexibility in retrievals and improved properties in an updating environment achieved by storing large arrays in 3+NF incurs a large key overhead (see for example, STIFFNESS MATRIX DEFINITION relation). Whether this overhead is worth while or not needs to be reviewed for each data relation.

The question as to whether graphics can be successfully integrated with such a scheme, as proposed, remains unanswered. Graphics files typically have list structures to achieve speed in display and rapid manipulation. Although graphics systems using relational and network data models have been implemented, performance remains an unknown.

(4) The Data Base Management System

In this section the MIMER DBMS is compared with Date's relational model presented in chapter IX and his discussion on System R and SQL.

(a) General. MIMER was flexible and easy to use. It dynamically

shared data from a consistent non-redundant store with many applications. It provided quick access independent of whether the data was accessed by selecting primary key values, secondary index values, or non-indexed field values. (Any number of indexes could be created and destroyed without affecting users, except in performance.) It also provided sequential file type access when no selection criteria were used. Selection of data fields to be retrieved was possible. Some integrity constraints were automatically provided within MIMER. MIMER also possessed a high level of machine independence.

It is claimed then that the use of a commercial DBMS represents a distinct step forward in data management over a system utilizing unrelated operating system files containing redundant data.

(b) Data Structures. MIMER data structures fell short of satisfying Date's relational model. Domains were not supported leaving the relationship between relations implicit. Primary key uniqueness was enforced and null values were prohibited in primary key fields. There was however no means of enforcing uniqueness in alternate key fields. Referential integrity was not enforced either, allowing reference to non-existent records.

Unlike System R and RIM (refer chapter XIV) variable length character strings were not allowed as field definitions. Code lists were also not supported. (A code list is a small finite set of values such as blue, green or red defined as a domain separate from the field definition.) Code lists could be implemented in MIMER by the user by defining another relation containing the code list values and referencing code values in the original relation by way of a foreign key attribute. In the particular implementation used, double precision real

data fields were not supported. This was a disappointing feature as it caused loss of precision in the finite element calculations.

Although not part of Date's relational model (but discussed in the literature), support for other data abstractions would provide considerable simplification and improve the relationship between the database model and the world of wagon design.

An example of these beneficial abstractions (which include a higher level of structure and operations allowed on this structure) would be the definition of vector, matrix and tensor types. This is not to say that 1NF is not desirable rather that it is also desirable for the user to have a direct way of storing, manipulating, and deleting these data types (as is possible for a string of characters).

Relational calculus (and algebra) operations should still be possible as should a view of the data as atomic attributes (because of the benefits of normal forms and because the user may wish to interrogate at the detailed array element level).

Generalisation or generic structure definition is another desirable extension. For example, BEAM ELEMENT is a FINITE ELEMENT, STIFFNESS MATRIX is an INTERPRETER VARIABLE and FLAT PLATE is a GEOMETRICAL ASSEMBLY. A generic structure would say the descendant relations must bear the same key domains as their ascendants allowing individual relations to be uniformly referred to. Every instance of a subordinate class has all the properties of the more general class.

Aggregation describes how component relations are part of a higher level aggregate entity. For example, STIFFNESS MATRIX DEFINITION is part of STIFFNESS MATRIX. For each instance of the aggregate (or hierarchical class) entity, there exists relations constituting its parts.

These abstractions are not essential elements of a DBMS but would allow a more elegant solution to some database design issues than was possible in this study using MIMER. If there were ways of defining these abstractions explicitly to the system, then the user would have the possibility of querying an entity as a whole rather than assembling different relations and thinking about all known relationships. For example, in order to delete a stiffness matrix in the prototype system using the Query Language at least two (and probably three) delete commands must be issued.

In MIMER there was no provision to specify an external view (except indirectly through MIMER/QL procedures). Other systems provide for definition in the data dictionary of a "view table". The view table is a table derived from one or more conceptual view tables and can be regarded as a window on the database. These view tables offer a degree of logical independence as they separate the external schema from the conceptual schema.

In MIMER, relations, relationships, and attributes could have been added without affecting users.

Although the MIMER database and MIMER data dictionary were integrated, the facilities offered in the data dictionary were lacking in areas such as documentation, project management, and database design tools.

(c) Data Manipulations. On the positive side, MIMER DML prohibited primary key updates and requests contained no reference to explicit access paths (thereby ensuring a high level of physical independence). In addition to the normal relational operators (such as equals and greater than), four operators ("begin with", "contain", and

their negations) were included for character type fields. MIMER did not eliminate duplicate rows with its retrieval command. (In the relational model the result of an operation is always a relation.)

If MIMER DML is compared with Date's report on SQL, deficiencies in usability features and in selective power of commands are evident in MIMER. In MIMER there was no ordering of retrieved fields, the rows in the database were sorted by primary key. A number of operators and quantifiers were absent in MIMER, for example, NOT, UNION, ANY, ALL and EXISTS. Arithemetic expressions involving fields were not valid in the select and where clauses. Multiple levels of nesting or subqueries were also not allowed. Built-in functions such as summation of values, average of values, maximum value, minimum value, group by field values, and "having" a condition for groups (similar to "where" for rows) were not available. There was also no way of specifying the null value in MIMER/QL.

In MIMER programming host language DML the value returned for a null value in the database depended on the field specification, that is, I2 (a two byte integer) null was different from I6 (a six byte integer) null value. Provision of a routine to test for null values or signalling null value in the routine return codes are two ways of improving this interface.

The programming host language DML was a separate language to MIMER/QL unlike in SQL programming host language DML where a precompiler translates the same SQL statements available as a query language into routine calls. In MIMER the interface was implemented through routine calls. For example, a series of calls was necessary to set up each select field. This approach results in program code that is not concise

and which tends to be unstructured. The system developer has more to learn and the same operations can not be tested in either environment. Such an approach is more interpretative than a compiled approach, resulting in slower execution.

MIMER portability could also be detrimental to efficiency.

(5) Encoding Language

The choice of a compatible subset of FORTRAN IV as the encoding language was costly in retrospect. For although compatible FORTRAN IV would have a high degree of portability, there were definite areas of weakness, for example, no character string facilities, limited data structuring, compulsory data typing, no error condition handling facilities, and no program initiation and linking facilities. Dynamic linking of programs has important influences on maximum program size, efficient use of memory and system architecture. Compulsory data typing led to inefficiencies because both the code stack and run-time stack had to be of either integer or real type. Type checking was performed by the prototype's code generator routines. Conversions between types and additional data structures added unnecessary CPU cycles to execution times.

A modern encoding language such as "C" does not have these deficiencies and has in addition multitasking, multitasking communications and data sharing facilities, and a full interface to the operating system.

Operating system procedural languages, such as PRIMOS CPL, also have these features but they are machine dependent and slow (because they are usually interpretive).

3. FURTHER DEVELOPMENT AND IMPLEMENTATION TASKS

This section proposes an outline implementation plan and discusses a number of areas for development but is not intended as an inexhaustive list of alterations and detailed developments.

(1) Further Work on the Implemented System

(a) Scope. There is much potential for expanding the analysis and procedural scope as well as the coverage of the database in wagon design. Apart from enabling more meaningful comparison with other approaches, the inclusion of more design tasks will enable the development of mechanisms for handling and storing as data a network of models and methods. It will enable the user to specify what is to be done (involving a number of tasks) at a higher level. A command at this level might be, for example:

```
SOLVE FEM FOR 'TARE' STRESSES WITH 'TARE' LOAD,
'KS' STRUCTURE USING STRUDL;
```

The database could be expanded by reconciling the design control and release features and the form and feature parametric models in the present system with manufacturing data (such as form and position tolerance, surface finish, surface treatment and heat treatment), more general descriptions of geometry (surfaces, shapes, and volumetric modelling, for example) and draughting entities. Abstracts, references, and so forth could be added to the database for technical papers, articles, catalogues, and trade literature.

The scope of the whole scheme could be increased to cover passenger

vehicles, locomotives, manufacturing and/or draughting.

In addition to textual entry and retrieval on the database, graphic reporting (or "plotting") could be an application which utilises a graphics subroutine library such as NCAR. Application programs themselves may use graphics in their user communication languages. Use of graphics in conjunction with the WAGON and GEOMETRY databanks opens up the possibility of a much better means of communication with the wagon designer.

(b) Information Processing Techniques. In addition to information processing in the form of programmed modules that operate solely on numerical data, other processing techniques that could be included are execution of decision tables, more general methods of equation solving, and symbolic manipulations.

Symbolic manipulations use symbols as values rather than numeric or boolean constants. Numerical methods or symbolic manipulations could be used for equation solving.

Decision tables provide a high level specification language for design standards and design office procedures that contain a lot of detailed logic. Methods would have to be found to store in the database both the decision table itself (that is, the relationship between the conditions and actions) and the relationship between instances of input data and output data. Processing methods would have to allow for the possibility of unknown condition values, at which point the decision table processing would have to suspend while data was sought.

The flexibility inherent in the prototype system in terms of methods and relationships used opens up possibilities in the use of probabilistic choice of course of action based, perhaps, on available

data and the desired data or goal. In the present system only deterministic decisions can be made on the path taken.

(c) Command Language. The present system is missing many features in its user environment. A HELP command could be added to give on-line information for both commands and variables, models, and methods. An ARGUMENTS command could display the list of variables in a specified model and/or method indicating which are dependent and which are independent.

For compound command steps interrupts and step by step monitoring could be implemented. Step tracing could display information on each step as it occurs, while variable tracing could display the value of variables when changed. Variables could be added or deleted from a watch list. As a soft interrupt, conditional break points or stop locations with actions to be performed on breaking execution could be introduced at the beginning and end of command steps. From these break points the wagon designer could opt to continue one command step at a time or continue until some condition is satisfied.

A printable session log could be made available to the user.

Spreadsheet type entry and display is another feature possible for processing with few data items. Screen mode editing type of entry, addition, and deletion for bulk tabular data with defaulting, repetition, generation rules and constants could also be possible.

Usability would be enhanced with user environment tailoring, incorporating such features as abbreviations; synonyms; renaming of variables, commands, and so on; user specification of defaults; range and validity checking; and controls over quantity of information in responses.

Fully annotated screen menus could be available for the novice and occasional user.

The scope of functions available in the command language could also be improved. Facilities could be added for users to edit and run procedures (perhaps specified in the form of a decision table). More language structures would have to be added to support the procedures feature. File manipulation commands (for example, to make logical connection to operating system file, to write formatted data, read procedures) could be implemented. The subset of operating system functions could be introduced so as to give file security and enable access control; integrity through file versioning, and so on; and to permit resource allocation, charging and scheduling.

Finally more character string operations and array variable types could be added together with array operations to add, zero, matrix invert, and so on.

(d) System Expansion. Programs could be developed to assist in the development of the system. For example, from a description of the commands to be added, the additions to the various command interpreter tables could be generated. In the same manner models, variables, and methods could be added to the interpreter and database; instructions could be added to the run-time interpreter; and decision tables and parameterised geometry could be added to the database.

Eventually the wagon designer could define and delete database variables and models. Methods could be related to stored procedures (written in the command language) or to decision tables developed by the wagon designer. At start up the interpreter could be initialised from the database improving integrity and if the state of the global area was

saved at session end, the wagon designer would commence the next session exactly where he left off. At this stage knowledge acquisition by the system is feasible.

(e) The Program. The program could benefit from alterations in a number of areas. For instance, larger instructions running as tasks, interrupt handling, storage of messages in tables, and improved message communication amongst modules.

(f) Benefit Measures. Systems measures such as CPU utilisation and disk accesses per minute are easier to quantify than such measures as response time, and ease of use. However surveys, questionnaires, productivity tests, and so forth, can be developed for these more difficult measures.

(2) Implementation Plan

The suggested implementation steps for NZR are:

(i) Audit functional specification.

(ii) Evaluate prototype integrated design system.

(iii) Revise specifications if necessary and develop benefit and cost measures.

(iv) Search for commercial products (including hardware) with which to implement specification.

(v) Determine for the basic system the balance between in house development and commercial products.

(vi) Plan pilot scheme which lasts six months to one year. Planning should include how to train staff, who to train, operational procedures, and so on, as well as software and hardware acquisition. Design models should be implemented in a staged manner after

consideration of at least the following factors: frequency of use, contribution to product quality, availability of parameter information, and development cost. Only a relatively small commitment (certainly less than one million dollars) to maintenance and development should have been made by this stage.

(vii) Implement pilot scheme, train staff, record benefit/cost, performance and utilisation data.

(viii) Analyse data, revise plan and action revised plan.

4. CONCLUDING REMARKS

This thesis set out to investigate ways of integrating the relevant published works, procedures used in NZR wagon design, and the design and operational experience of NZR staff. Computers were to aid the designer in this integrated scheme.

The thesis contains the results of a literature survey, a survey of NZR Design Office procedures and documentation, and staff interviews. This information was used to produce a functional specification of a computer based integrated scheme. No existing system was found that fulfilled the major parts of the specification. A prototype system was developed to test some aspects of the specification.

The prototype represents a first attempt to use a commercial relational database management system with existing design models and solution procedures. It could be regarded as a designers "computerised notebook". It offers the designer substantial flexibility in the use of data and in the direction in which he can proceed but maintains integrity by storing the relationships between data.

A more complete system would reduce the amount of relearning and the repetitive and error prone re-entering of data when different analysis models and methods are used. It should also be a system in which advances in knowledge are more easily accommodated. It should not restrict the purposeful behaviour of the wagon designer as much as some previous computer aids.

Now a closer integration in design than presently available is possible which offers a positive contribution to an organisation's objectives.

ACKNOWLEDGEMENTS

The author is grateful to many people for providing advice, information, and encouragement. The author is particularly grateful to Professor H. McCallion, the author's supervisor, for his advice and guidance throughout the work. The author is also grateful to NZR for allowing much of the work to be done using company time and resources. The author would like to express his appreciation to the staff of NZR for the tremendous co-operation, encouragement, and patience they have shown throughout this work. Finally, the author would also like to thank his wife and family for their support and encouragement.